

Hybrid parallel programming & advanced topics

Simon Scheidegger
simon.scheidegger@unil.ch
September 27th, 2018
Cowles Foundation – Yale

Including adapted teaching material from books, lectures and presentations by
B. Barney, B. Cumming, W. Gropp, G. Hager, M. Martinasso, R. Rabenseifner, O. Schenk, G. Wellein

Today's Roadmap:

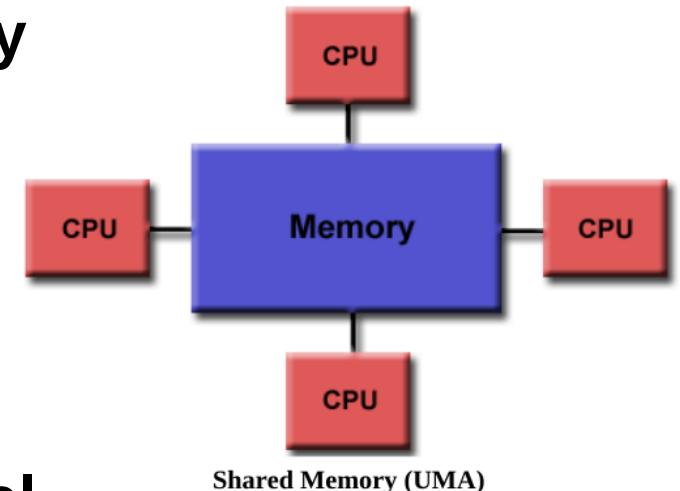
1. Hybrid parallelism.
2. Hybrid parallel Dynamic Programming.
3. Libraries & Advanced topics.

Reminder: Shared memory systems

- Process can access same **GLOBAL** memory

- **Uniform Memory Access (UMA)** model

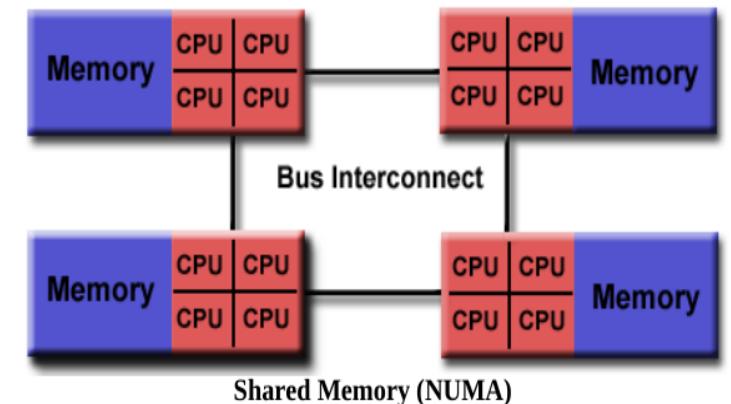
- Access time to memory is uniform.
- Local cache, all other peripherals are shared.



- **Non-Uniform Memory Access (NUMA) model**

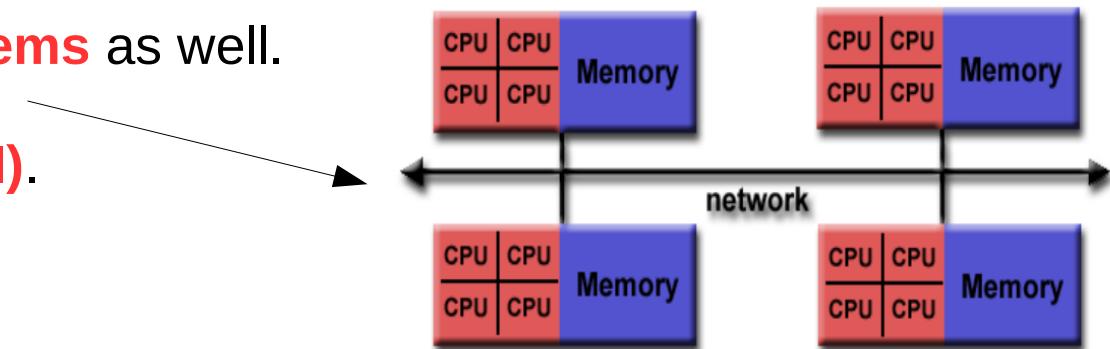
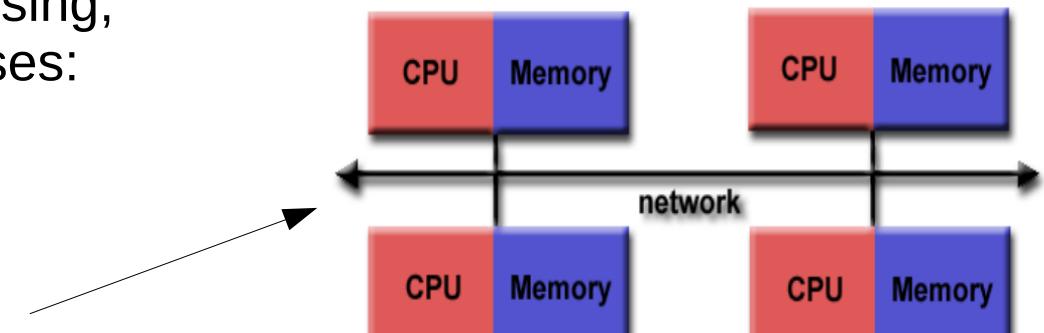
- Memory is physically distributed among processors.
- **Global virtual address spaces accessible from all processors.**
- Access time to local and remote data is different.

→ **OpenMP**, but other solutions available (e.g. Intel's TBB).



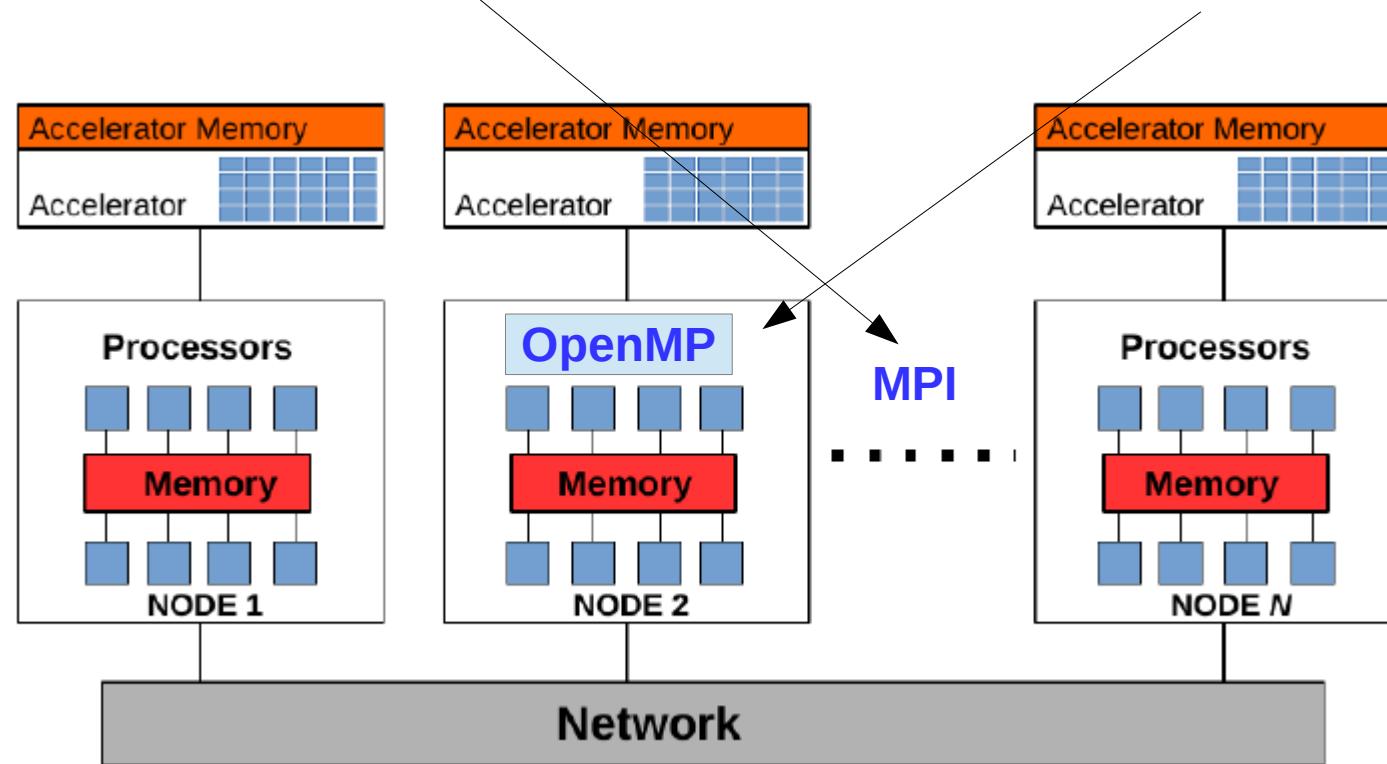
Reminder: Distributed-memory systems

- We need to use explicit message passing, i.e., communication between processes:
 - Most tedious and complicated but also the most flexible parallelization method.
- Message passing is required if a parallel computer is of **distributed-memory** type, i.e., if **there is no way for one processor to directly access the address space of another**.
- However, it can also be regarded as a programming model and used on shared-memory or **hybrid systems** as well.
- **Message Passing Interface (MPI)**.



Today's HPC systems

- Efficient programming of clusters of **shared memory nodes**
- Hierarchical system layout
- Hybrid programming seems natural:
 - **MPI among the nodes.**
 - **Shared memory programming inside of each node – OpenMP.**



Hybrid parallelism with MPI and OpenMP

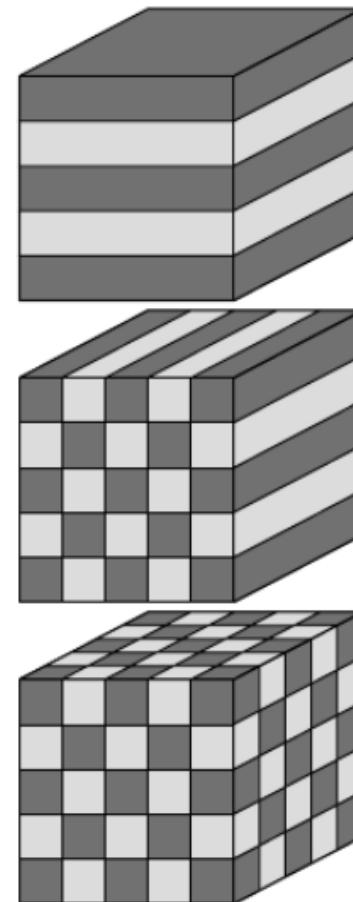
When Does Hybridization Make Sense?

- When one wants to scale a shared memory OpenMP application for use on **multiple SMP nodes** in a cluster.
- When one wants to **reduce an MPI application's sensitivity to becoming communication bound**.
- **When one is designing a parallel program (nowadays) from the very beginning.**
- for 8/16/32/64/...ranks per multi-core node, this can have scaling problems with many nodes/MPI ranks.

Hybridization Using MPI and OpenMP

- facilitates cooperative shared memory (OpenMP) programming across clustered SMP nodes.
- MPI facilitates **communication** among SMP nodes.
- OpenMP manages the **workload** on each SMP node.
- **MPI and OpenMP are used in tandem** to manage the overall concurrency of the application.

Domain decomposition

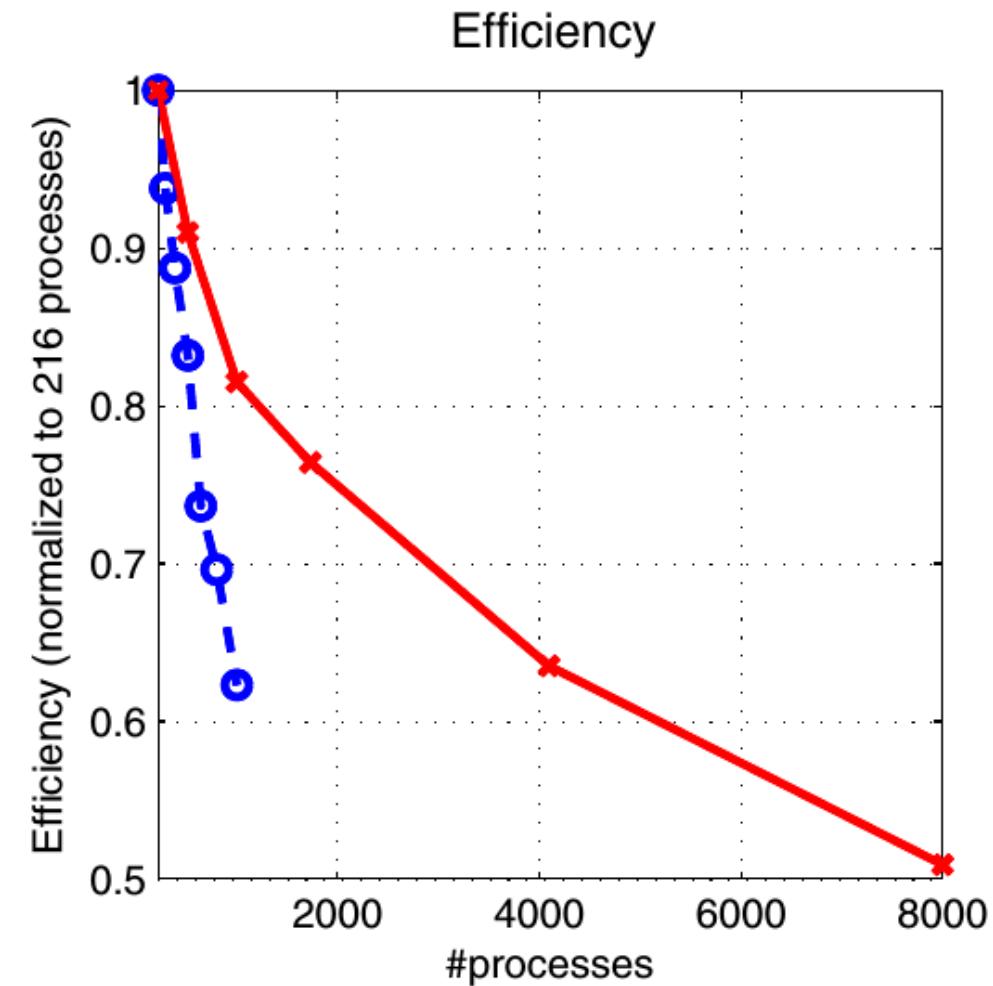
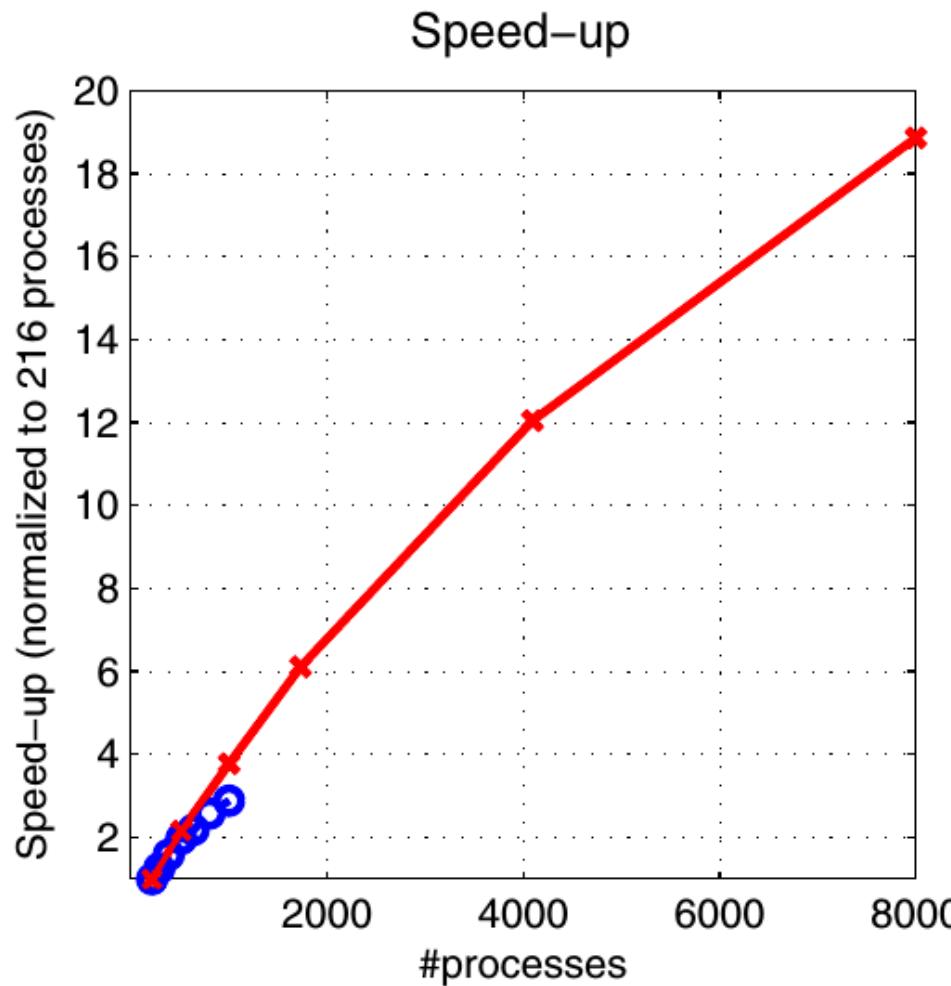


Example: Scaling “MPI” vs. “Hybrid”

See, e.g., Käppeli et al. (2011) <http://iopscience.iop.org/article/10.1088/0067-0049/195/2/20/pdf>

Hybrid parallelization for cubic domain decomposition:

- Reduces the amount of memory consumption.
- Reduces the amount of communication required.



Memory consumption & mapping

- Memory consumption MPI & OpenMP with n threads per MPI process:
 - Duplicated data may be reduced by factor n .
- How many threads per MPI process?
SMP node = with m sockets (NUMA domains) and n cores/socket
- How many threads (i.e., cores) per MPI process?
 - Too few threads, too much memory consumption
- Optimum:
 - somewhere between 1 and $m \times n$ threads per MPI process.
- Typical optima:
 - 1 MPI process per whole SMP node.
 - 1 MPI process per socket.
 - 2 MPI processes per socket.

Mapping (1)

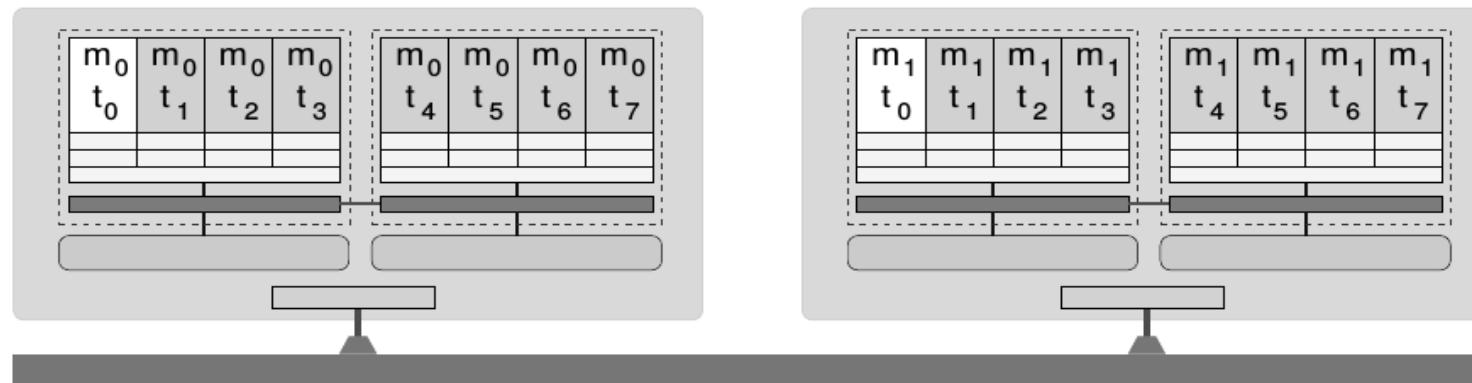


Figure 11.3: Mapping a single MPI process with eight threads to each node.

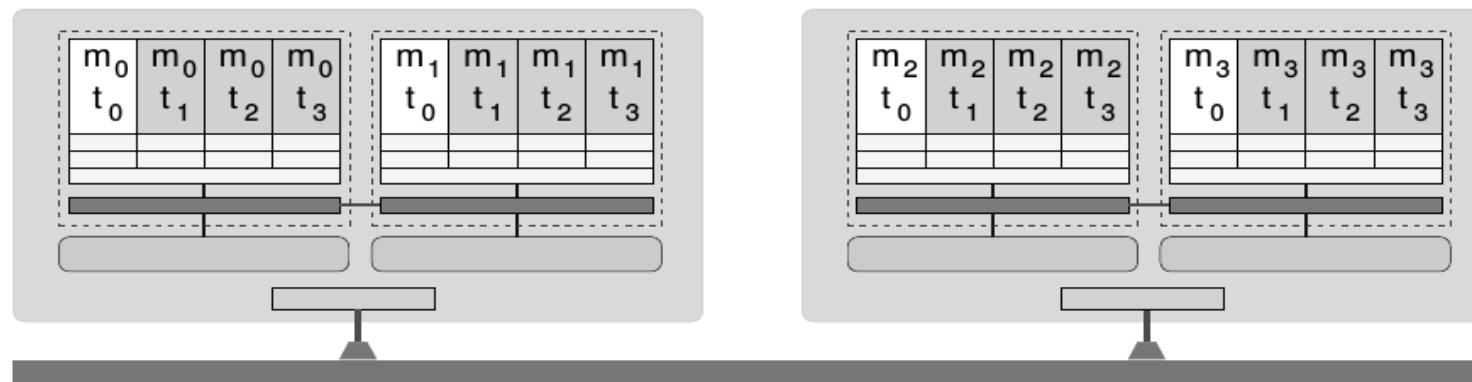


Figure 11.4: Mapping a single MPI process with four threads to each socket (L3 group or locality domain).

Mapping (2)

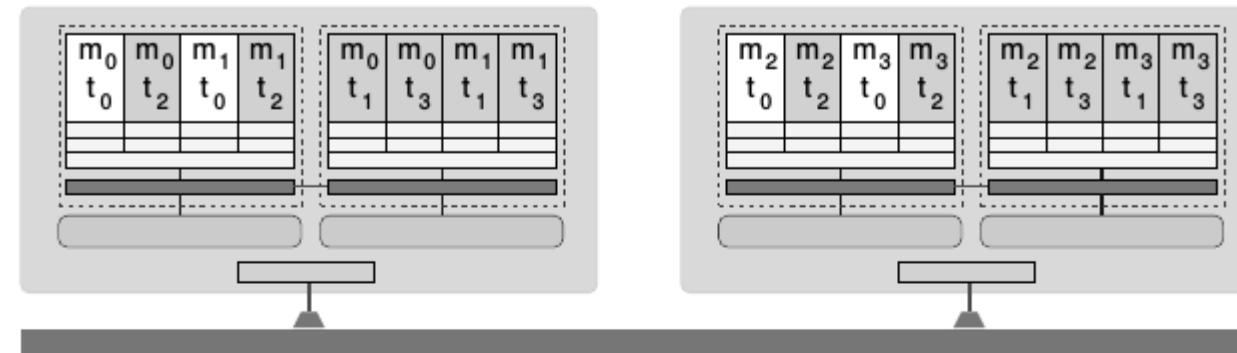


Figure 11.5: Mapping two MPI processes to each node and implementing a round-robin thread distribution.

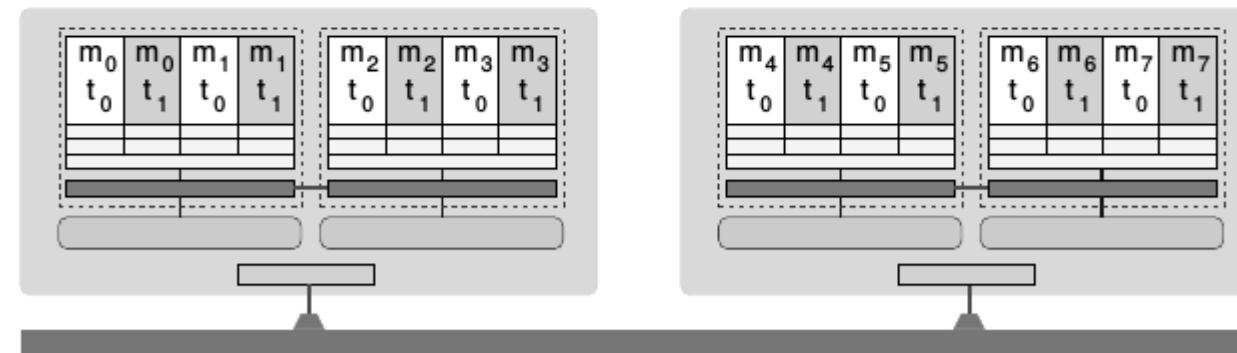


Figure 11.6: Mapping two MPI processes with two threads each to a single socket.

Opportunities

Algorithmic opportunities due to larger physical domains inside of each MPI process:

- If MPI domain decomposition is based on physical zones:
Nested Parallelism Outer loop with MPI / inner loop with OpenMP.
- **Load-Balancing**: Using OpenMP dynamic and guided work sharing
- **Memory consumption**: Significantly reduction of replicated data on MPI level.
- **Reduced MPI scaling problems**: Significantly reduced number of MPI processes
- Opportunities, if MPI speed-up is limited due to algorithmic problem
- **Significantly reduced number of MPI processes**.

A common way to implement hybrid parallelism

Fortran

```
include 'mpif.h'  
program hybsimp  
  
call MPI_Init(ierr)  
call MPI_Comm_rank (...irank,ierr)  
call MPI_Comm_size (... isize,ierr)  
! Setup shared mem, comp. & Comm  
  
 !$OMP parallel do  
 do i=1,n  
   <work>  
 enddo  
 ! compute & communicate  
  
 call MPI_Finalize(ierr)  
end
```

C/C++

```
#include <mpi.h>  
int main(int argc, char **argv){  
 int rank, size, ierr, i;  
  
 ierr= MPI_Init(&argc,&argv[]);  
 ierr= MPI_Comm_rank (...,&rank);  
 ierr= MPI_Comm_size (...,&size);  
 //Setup shared mem, compute & Comm  
  
 #pragma omp parallel for  
 for(i=0; i<n; i++){  
   <work>  
 }  
 // compute & communicate  
  
 ierr= MPI_Finalize();
```

Example

```
#include <stdio.h>
#include "mpi.h"
#include <omp.h>

int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    int iam = 0, np = 1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    #pragma omp parallel default(shared) private(iam, np)
    {
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
        printf("Hello from thread %d out of %d from process %d out of %d\n",
               iam, np, rank, numprocs);
    }

    MPI_Finalize();
}
```

MPI

OpenMP

Example: "hello world hybrid"

1. go to YaleParallel2018/day4/code/hybrid:

```
> cd YaleParallel2018/day4/code/hybrid
```

2. Have a look at the code

```
> vi 1a.hello_world_hybrid.cpp
```

3. compile by typing:

```
> source setupenv.sh (load correct MPI library)
```

```
> make
```

```
mpicxx -fopenmp 1a.hello_world_hybrid.cpp -o 1a.hello_world_hybrid.exec
```

4. Experiment with different numbers of threads/MPI Processes

```
> export OMP_NUM_THREADS=4
```

```
> mpirun -np 2 ./1a.hello_world_hybrid.exec
```

Slurm & Hybrid Jobs on GRACE

```
#!/bin/bash
#SBATCH --job-name=hybrid
#SBATCH --output=hydrid_job.txt
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=5
#SBATCH --nodes=2
#SBATCH --time=10:00
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
mpirun hello_hybrid.mpi
```

- **8 MPI** processes, distributed on **2 nodes**.
- **5 threads per MPI process**
- $8 \times 5 \text{ CPUs} = 40 \text{ CPUs}$

Example: Slurm on GRACE

1. go to YaleParallel2018/day4/code/hybrid:

```
> cd YaleParallel2018/day4/code/hybrid
```

2. Have a look at the code

```
> vi submit_hybrid_yale.sh
```

3. compile by typing:

```
> make
```

4. Experiment with different numbers of nodes/threads/MPI Processes and look at the output.

```
> sbatch submit_hybrid_yale.sh
```

Putting things together: Hybrid parallel DP



recall – the model

$$V_{new}(k, \Theta) = \max_c (u(c) + \beta \mathbb{E}\{V_{old}(k_{next}, \Theta_{next})\})$$

$$\text{s.t. } k_{next} = f(k, \Theta_{next}) - c$$

$$\Theta_{next} = g(\Theta)$$

States of the model:

- k : today's capital stock → **There are many independent k 's**
- Θ : today's productivity state → **The Θ 's are independent**

Choices of the model:

- k_{next}

→ k , k_{next} , Θ and Θ_{next} are limited to a finite number of values

solver.cpp – the critical loops

```
for (int itheta=0; itheta<ntheta; itheta++) {           2). *split MPI communicator
    /*
        Given the theta state, we now determine the new values and optimal policies corresponding to each
        capital state.
    */

    for (int ik=0; ik<nk; ik++) {                         1). distribute k's via OpenMP & MPI
        // Compute the consumption quantities implied by each policy choice
        c=f(kgrid(ik), thetagrid(itheta))-kgrid;

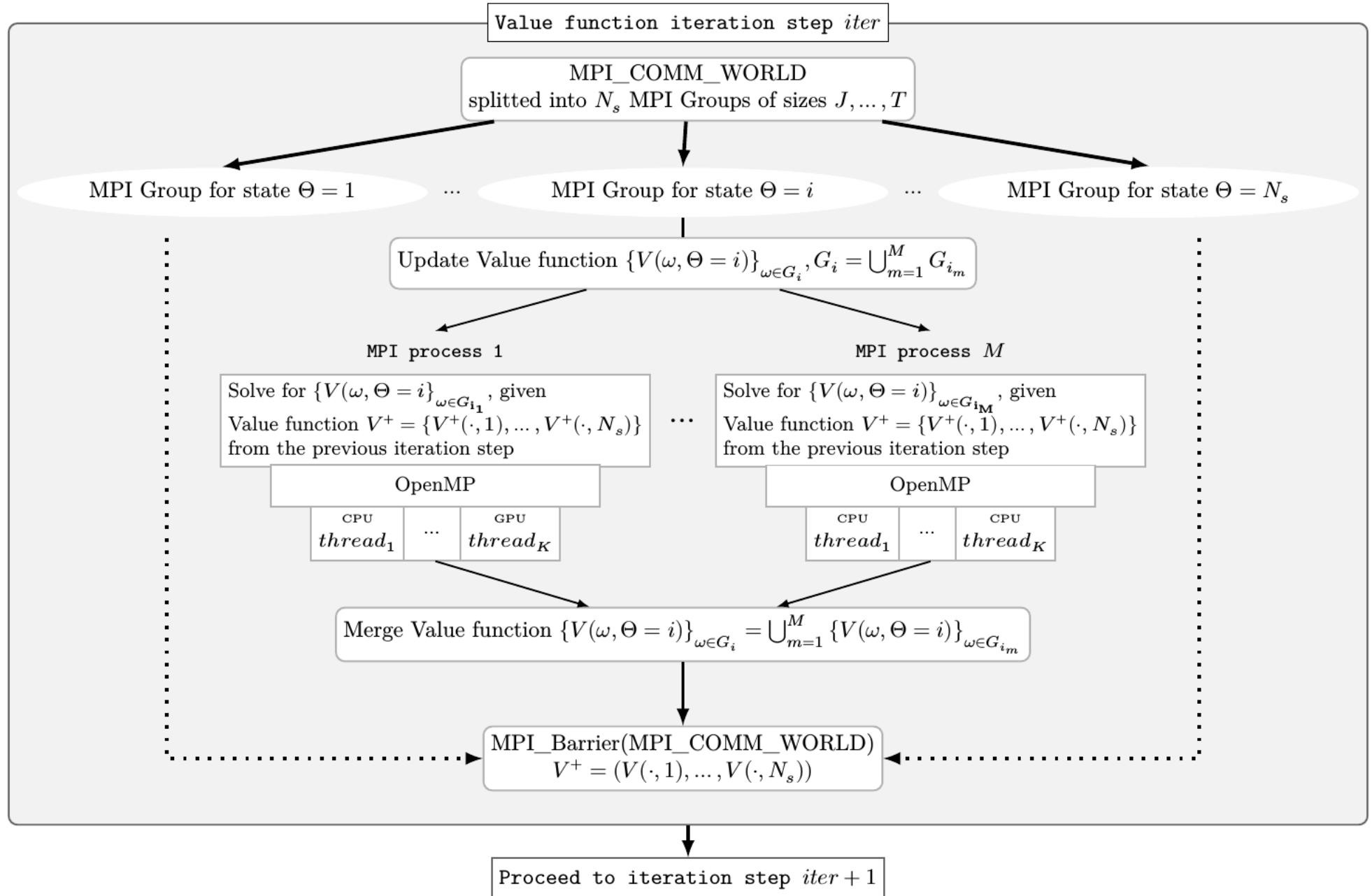
        // Compute the list of values implied by each policy choice
        temp=util(c) + beta*ValOld*p(thetagrid(itheta));

        /* Take the max of temp and store its location.
         * The max is the new value corresponding to (ik, itheta).
         * The location corresponds to the index of the optimal policy choice in kgrid.
         */
        ValNew(ik, itheta)=temp.maxCoeff(&maxIndex);

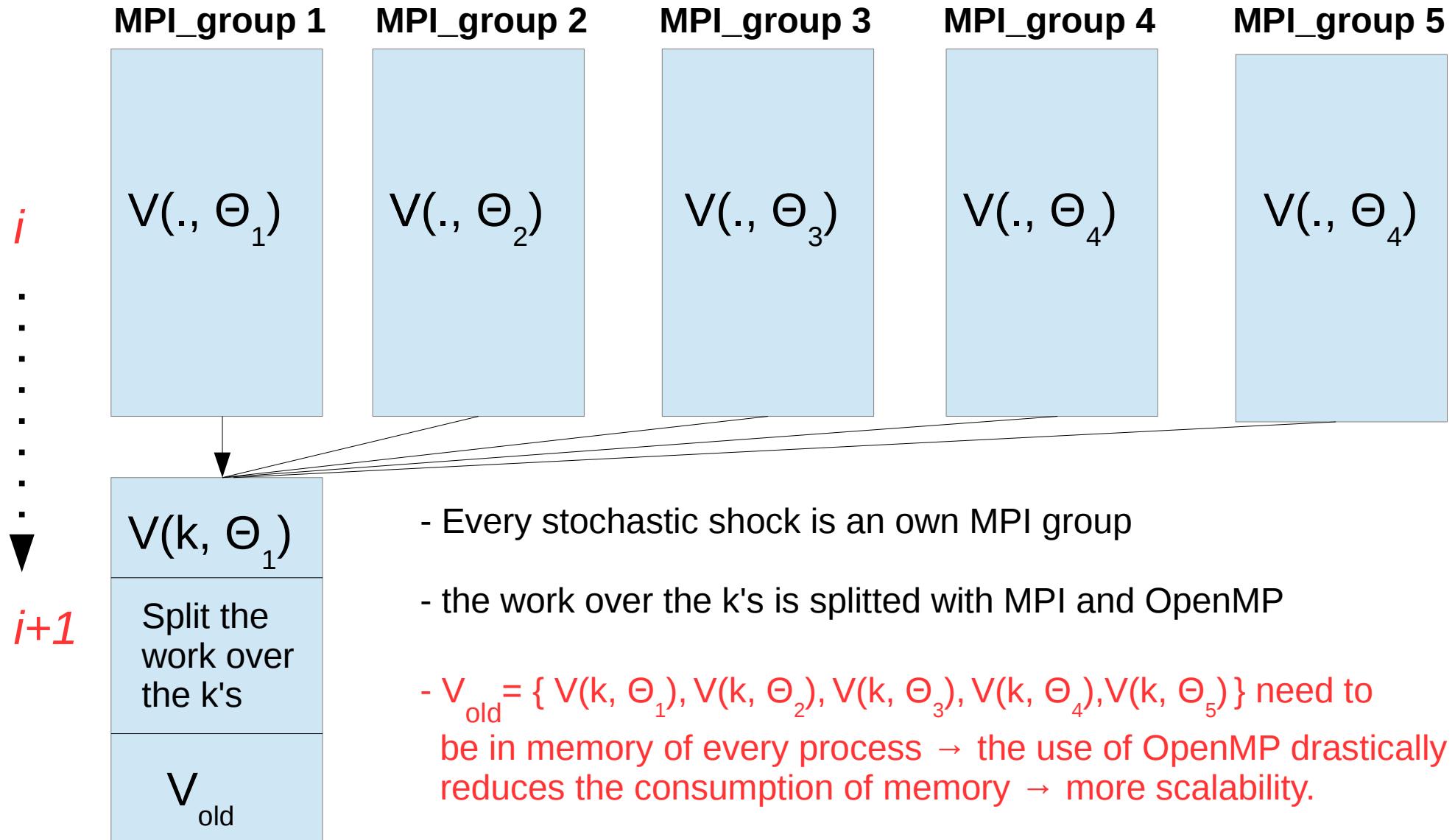
        Policy(ik, itheta)=kgrid(maxIndex);
    }
}
```

loops to worry about

The parallelization scheme



The parallelization scheme



Let's look at the code together!

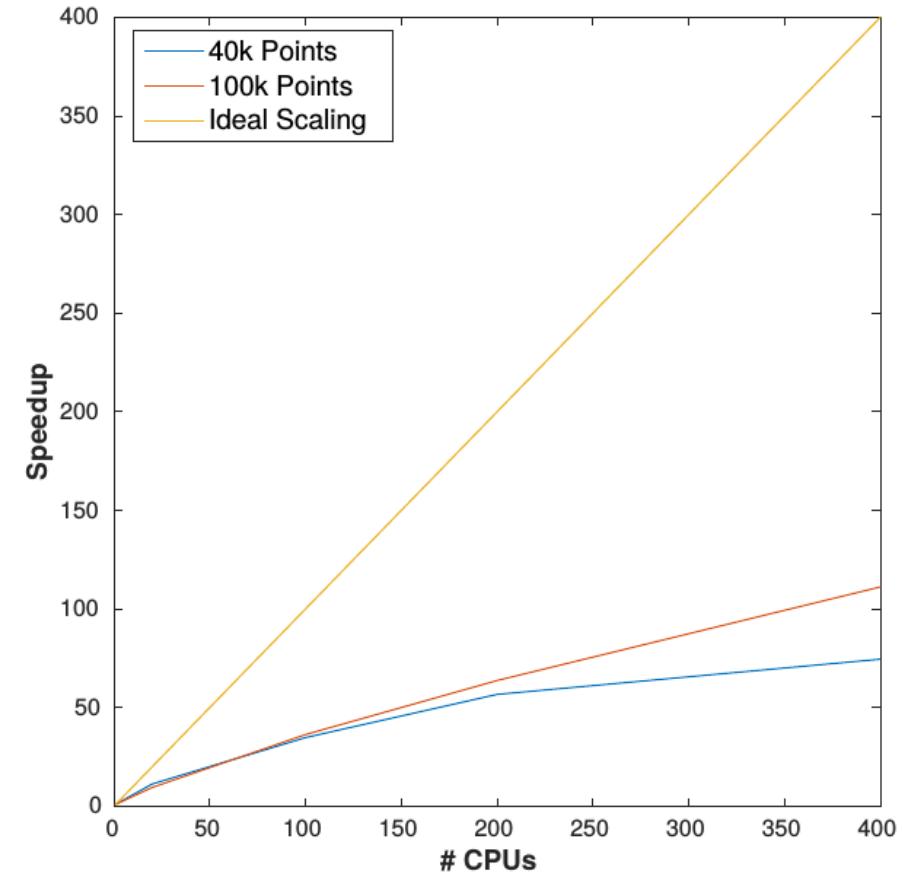
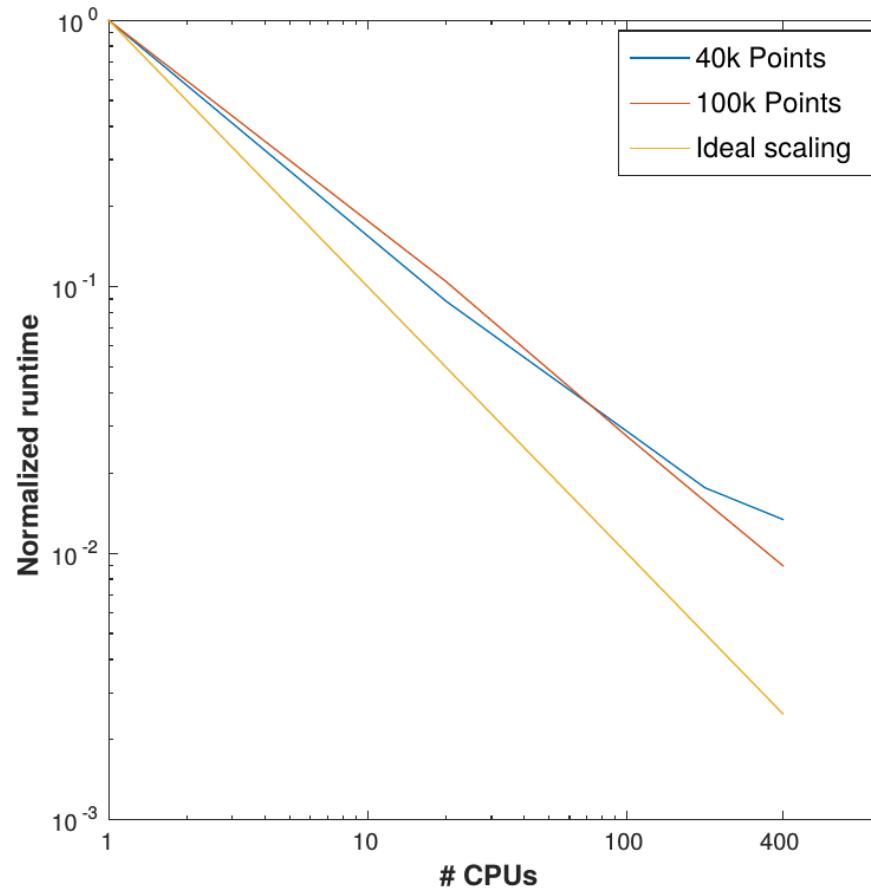
1) OMP & MPI:

`YaleParallel2018/day4/code/DynamicProgramming/DP_hybrid`

2) MPI groups:

`YaleParallel2018/day4/code/DynamicProgramming/DP_MultComms`

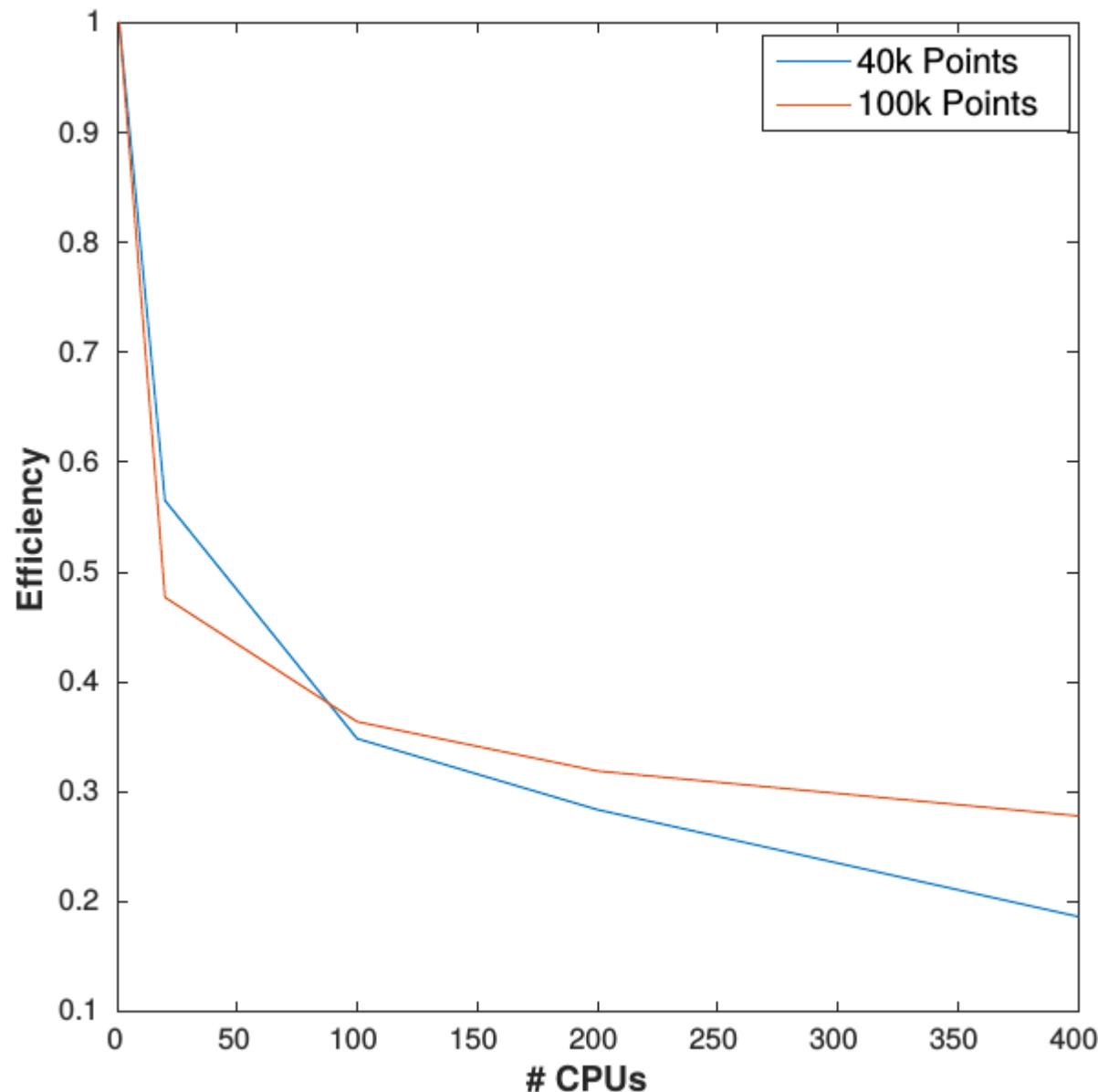
Performance: 40 & 100k points



Scalability for this problem size is limited.

- Overhead quickly becomes dominant: the runtime is ~3 sec for 400 CPUs with 40k points.
- **Appropriate baseline is probably rather 1 single node**

Efficiency: 40 & 100k points



Libraries & Advanced topics

1. Libraries
2. Emerging hardware
3. Trending topics

Libraries*



*The other commandments got lost in translation

What is a software library?

- A set of related functions to accomplish tasks required by more than one application.
- Written and used by different people.
- Relies on an Application Programming Interface (API).
- Typically versioned, documented, distributed, licensed.

Libraries for Scientific Computing: Pros

- **Don't reinvent the wheel.**
- **Don't reimplement the wheel.**
- **Use the wheel!**
- Leverage the work of experts.
- **Focus on your part of the “stack” to do science.**
- Experiment quickly.
- **Avoid “lock in”** with respect to data structures and algorithms (maybe a wheel wasn't the right choice).
- Open source or community projects allow consolidation of efforts from many people.
- **Continuity** on time scales longer than projects/PhDs/grants/careers.
- Collaborative efforts good for science.

Libraries for Scientific Computing: Cons

- Learning curves.
- Versioning, changing APIs.
- Bugs that someone else must fix.
- Syntax, design choices.
- Lack of documentation (or local experts).
- Oversold software.
- The scientific risks of using algorithms (or hardware) that you don't understand.

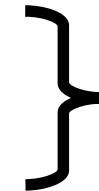
Where libraries show up

There are many libraries for scientific computing. Some are specialized to **certain application** areas, others are **quite general**.

- Linear Algebra Libraries
- Sparse Linear Algebra
- Iterative Solvers
- Eigensolvers
- GPU-enabled Linear Algebra Libraries
- SNOPT
- IPOPT



Examples for general libs.



Examples for Specialized libs.

Dense Linear Algebra

- Dense linear algebra, that is – linear algebra on matrices that are stored as two-dimensional arrays has been standardized for a considerable time.
- You almost certainly use these operations already.
- You likely leverage (perhaps indirectly) libraries to do so.
 - Typical Operations include elementary **element-wise operations on matrices and vectors** : $A + B$,etc.
 - Norms, inner products, matrix-matrix multiplies, matrix-vector multiplies : $\|x\|_2$, $\langle x, y \rangle$, AB , Ax ,
 - Cholesky factorization: $A = LL^T$, L lower triangular
 - QR decomposition: $A = QR$, $Q^H Q = I$, R upper triangular
 - LU factorization: $A = P^T LU$, P permutation, L lower triangle, R upper triangle
 - Triangular solves $y = L^{-1}x$
 - Eigenvalue decomposition: $Ax = \lambda x \iff A = Q\Lambda Q^T$, $Q^H Q = I$
 - Singular value decomposition: $A = U\Sigma V^H$, $U^H U = I$, $V^H V = I$

BLAS

- The basic operations are defined by the three levels of **Basic Linear Algebra Subprograms (BLAS)**:
 - **Level 1** defines **vector operations** that are characterized by a single loop.
 - **Level 2** defines **matrix vector** operations, both explicit such as the matrix-vector product, and implicit such as the solution of triangular systems.
 - **Level 3** defines **matrix-matrix operations**, most notably the matrix-matrix product.
- The name ‘BLAS’ suggests a **certain amount of generality**, but the original authors were clear that these sub-programs only covered dense linear algebra.
- Attempts to standardize sparse operations have never met with equal success.

BLAS & Lapack

- Fundamental numerical libraries.
- Many implementations, optimized for different architectures.

- BLAS

- vector operations (BLAS-1)
- matrix-vector operations (BLAS-2)
- matrix-matrix operations (BLAS-3)

LAPACK

- Matrix factorization and linear system solution
- Least squares

SCALAPACK

- distributed memory LAPACK

- **Available implementations** on compute clusters often include the following:
 - Intel's math kernel library (MKL) includes BLAS and LAPACK,
 - Cray's libsci : heavily optimized BLAS, LAPACK, SCALAPACK e.g. within the Cray, PGI, and GNU environments.

Example for GPUs: MAGMA

<http://icl.cs.utk.edu/magma/>

The screenshot shows the official website for MAGMA (Matrix Algebra on GPU and Multicore Architectures). The header features a large, stylized "MAGMA" logo in red and white. Below the header is a navigation menu on the left with links to Home, Overview, News, Downloads, Publications, People, Partners, Documentation, and User Forum. The main content area has two columns. The left column contains a brief description of the project's goal: "The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems." It also includes a paragraph about the research idea and a note to refer to publications for more information. The right column is titled "Latest MAGMA News" and lists several news items with dates and titles, such as "MAGMA 2.0 released" (2016-02-09), "MAGMA 2.0 beta released" (2016-01-06), "MAGMA 1.7.0 released" (2015-09-15), "MAGMA MIC 1.4.0 for Intel Xeon Phi Coprocessors Released" (2015-07-12), and "MAGMA 1.6.2 released" (2015-05-05). At the bottom, there is a logo for ICL (Institute for Computational Mathematics) and a row of logos for sponsors: U.S. Department of Energy, NNSA, Industry Support From: AMD, intel, The MathWorks, and NVIDIA.

MAGMA

Home
Overview
News
Downloads
Publications
People
Partners
Documentation
User Forum

Matrix Algebra on GPU and Multicore Architectures

The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.

The MAGMA research is based on the idea that, to address the complex challenges of the emerging hybrid environments, optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, we aim to design linear algebra algorithms and frameworks for hybrid manycore and GPU systems that can enable applications to fully exploit the power that each of the hybrid components offers.

Please use any of the following publications to [reference MAGMA](#).

Latest MAGMA News

2016-02-09 [MAGMA 2.0 released](#)
2016-01-06 [MAGMA 2.0 beta released](#)
2015-09-15 [MAGMA 1.7.0 released](#)
2015-07-12 [MAGMA MIC 1.4.0 for Intel Xeon Phi Coprocessors Released](#)
2015-05-05 [MAGMA 1.6.2 released](#)

ICL

Sponsored By: Industry Support From:

Sparse Linear Algebra

For a tutorial see <http://www.users.csbsju.edu/~mheroux/ISC2016HerouxTutorial.pdf>

- Use cases: **sparse PDEs, big sparse data.**
- Fundamentally very different from dense linear algebra; operations are difficult to vectorize.
- Typically limited by data movement (memory bandwidth), not floating-point performance.
- Any operator which can be applied (hence potentially inverted) in linear time must be sparse, and most sparse linear algebra libraries are aimed at large systems.
- Efficient for repeated solves suboptimal scaling and entry-dependent factorization time and storage.
- Challenging to parallelize.
- For large-enough systems, eventually beaten by optimally-scaling methods (iterative and/or multilevel algorithms).

Example: <http://www.pardiso-project.org/>

Download the package
(licence & binary) from there.

You want e.g.

libpardiso600-GNU720-X86-64.so

Add the binary here:

YaleParallel2018/day4/code_day4/pardiso_example/lib

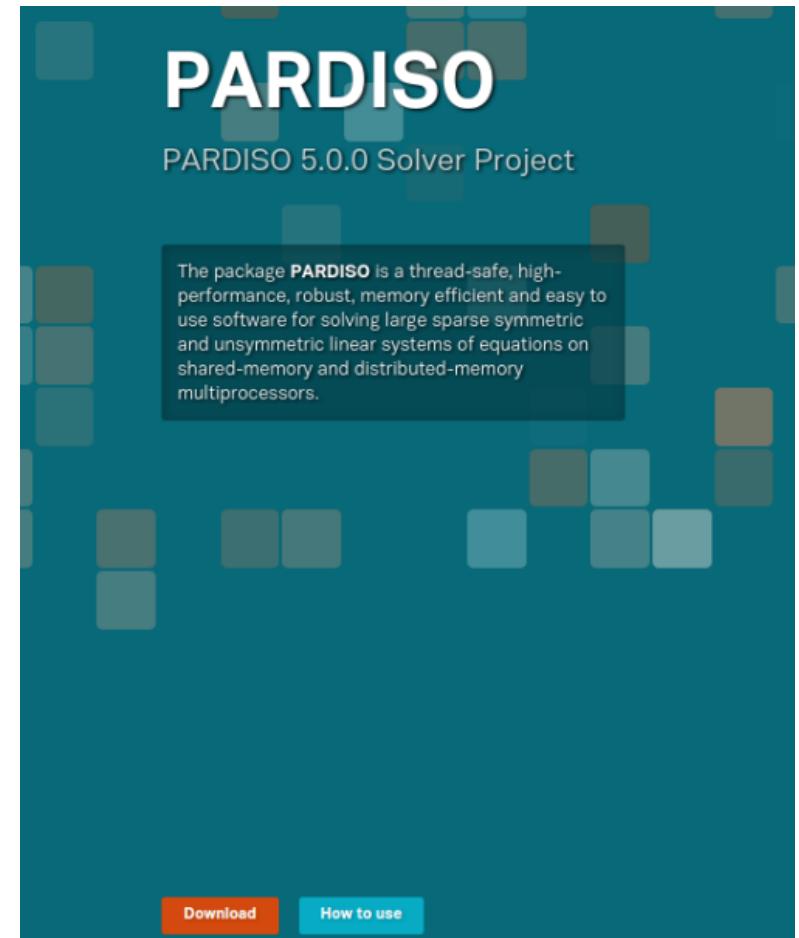
Copy the licence key to a
file named pardiso.lic

Make the licence visible by

a) adding it to your .bashrc

e.g. export PARDISO_LIC_PATH="~/licences/pardiso.lic"

b) putting pardiso.lic to your home directory (cp pardiso.lic ~)



What hardware do you have?

Current Available Libraries

Version 5.0.0 (Architecture x86-64, 64-bit)

Compiler	Operating System	PARDISO Libraries
gcc/gfortran 4.6.1	Linux	libpardiso500-GNU461-X86-64.so
gcc/gfortran 4.6.3	Linux	libpardiso500-GNU463-X86-64.so
gcc/gfortran 4.7.2	Linux	libpardiso500-GNU472-X86-64.so
icc/ifort 13.0.1	Linux	libpardiso500-INTEL1301-X86-64.so
mpicc/mpif77 4.5.0	Linux, MPI	libpardiso500-MPI-GNU450-X86-64.so
mpicc/mpif77 4.6.1	Linux, MPI	libpardiso500-MPI-GNU461-X86-64.so
mpicc/mpif77 4.6.3	Linux, MPI	libpardiso500-MPI-GNU463-X86-64.so
mpicc/mpif77 4.7.2	Linux, MPI	libpardiso500-MPI-GNU472-X86-64.so
icc/ifort 13.0.1	Linux, MPI	libpardiso500-MPI-INTEL1301-X86-64.so
icc/ifort 10.1	Windows (with optimized BLAS/LAPACK from Intel MKL)	libpardiso500-WIN-X86-64.dll, libpardiso500-WIN-X86-64.lib
gcc/gfortran 4.7.1	MAC OSX 10.6.5	libpardiso500-MACOS-X86-64.dylib, libomp5.dylib

Other libraries can be compiled upon request.

Please let us know in case that some of these libraries are not working for you.

1-dimensional Poisson equation

- Let us assume a discrete Poisson equation in 1D Cartesian coordinates

$$\frac{d^2u}{dx^2} = f(x), \quad x \in [0, 1] \quad u(0) = u_0 \quad \text{and} \quad u(1) = u_1,$$

- Let's discretize it, having m stencils ($i \in [1, m]$), and boundary values at $i = 0$ and $i = m + 1$.
- At $i = 1$, the Poisson equation then reads: $\Phi_0 - 2\Phi_1 + \Phi_2 = 4\pi G h^2 \rho_1$

The lower boundary condition Φ_0 is now shifted to the other side of the equation:

$$-2\Phi_1 + \Phi_2 = 4\pi G h^2 \rho_1 - \Phi_0$$

The upper boundary at $i = m$ is treated in analogy: $\Phi_{m-1} - 2\Phi_m + \Phi_{m+1} = 4\pi G h^2 \rho_m - \Phi_{m+1}$

Hence, the Poisson equation can be casted into a matrix notation, representing an $m \times m$ linear system of the general form

$A\vec{\Phi} = \vec{\rho}.$

Multi-dimensional Poisson equation

- In 3D Cartesian coordinates, the Poisson equation for the gravitational potential reads:

$$\Delta\Phi(x, y, z) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \Phi = 4\pi G\rho(x, y, z)$$

- The discretized version of this equation (assuming a uniform spatial discretization) on an **m × n × k grid** yields the following formula:

$$\begin{aligned} \Phi_{i-1,j,k} + \Phi_{i,j-1,k} + \Phi_{i,j,k-1} - 6\Phi_{i,j,k} + \\ \Phi_{i+1,j,k} + \Phi_{i,j+1,k} + \Phi_{i,j,k+1} = 4\pi G h^2 \rho(x, y, z), \end{aligned}$$

- **h** is the grid spacing.
- This can again be casted into a matrix notation, representing an **mnk × mnk linear system** of the general form

$$A\vec{\Phi} = \vec{\rho}.$$

Sparse matrix

For a $3 \times 3 \times 3$ ($m = 3$, $n = 3$, $k=3$) grid with all the boundary nodes fixed (set to be zero), the matrix A of the system would look as displayed left below.

In our notation, the entries of the vectors Φ and ρ are defined as below right.

$$\vec{\Phi} = \begin{pmatrix} \Phi_{111} \\ \cdot \\ \cdot \\ \cdot \\ \Phi_{11k} \\ \Phi_{121} \\ \cdot \\ \cdot \\ \cdot \\ \Phi_{1nk} \\ \Phi_{211} \\ \cdot \\ \cdot \\ \cdot \\ \Phi_{mnk} \end{pmatrix}; \quad \vec{\rho} = 4\pi G h^2 \begin{pmatrix} \rho_{111} \\ \cdot \\ \cdot \\ \cdot \\ \rho_{11k} \\ \rho_{121} \\ \cdot \\ \cdot \\ \cdot \\ \rho_{1nk} \\ \rho_{211} \\ \cdot \\ \cdot \\ \cdot \\ \rho_{mnk} \end{pmatrix}$$

This (and many other) PDEs can be solved by inverting the sparse matrix

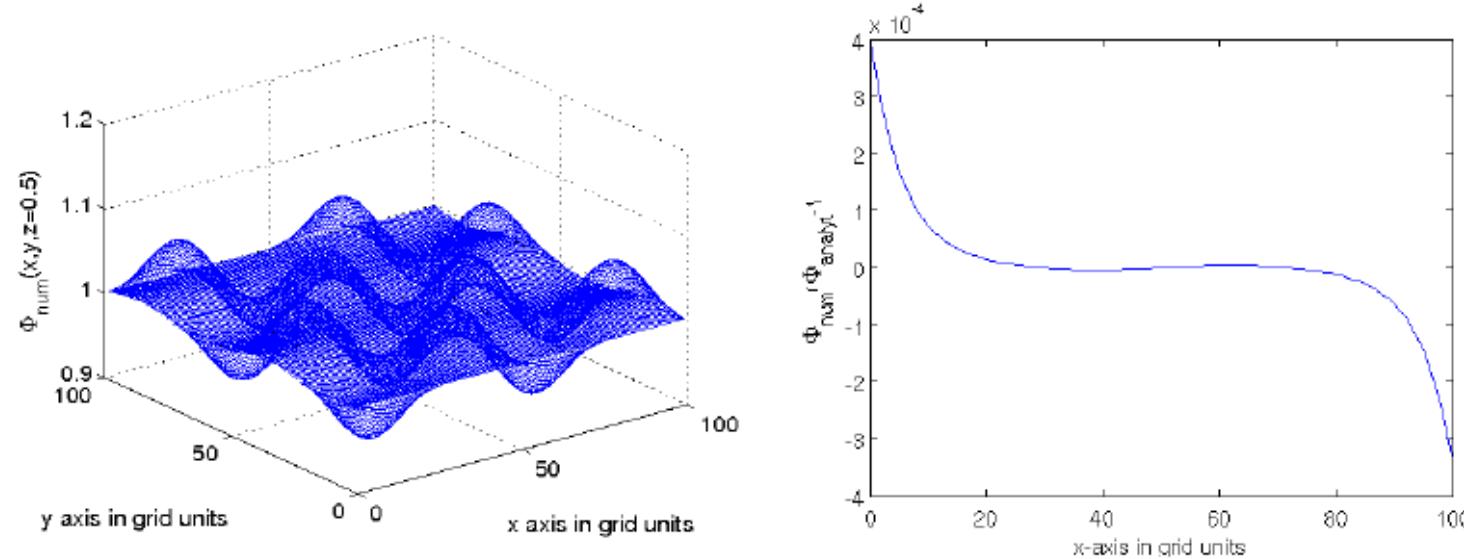
An analytical example

One exemplary analytical test: $\Delta\Phi(x, y, z) = f(x, y, z) = -48\pi^2 \sin(4\pi x) \sin(4\pi y) \sin(4\pi z)$.

Let's map the interval $x, y, z \in [0, 100]^3$, mapped onto grid points

We impose Dirichlet boundary conditions $\partial\Omega = 1$.

The analytical solution reads $\Phi(x, y, z) = \sin(4\pi x) \sin(4\pi y) \sin(4\pi z)$



Left panel: Numerical solution of eq. 2.85 in the $z = 0.5$ plane. **Right panel:** Relative error $\Phi_{num}(x, 0.5, 0.5)/\Phi_{analyt}(x, 0.5, 0.5) - 1$ along the x-axis.

Example

0. Prepare environment

```
module load Libs/GCC/5.2.0  
module load Libs/netlib  
module Langs/GCC/5.2.0
```

1. Go to

```
> cd YaleParallel2018/day4/code_day4/pardiso_example
```

2. Have a look at the code

```
>vi driver.f90
```

3. compile by typing:

```
> source setenv_yale.sh  
> make
```

4. run the code

```
>export OMP_NUM_THREADS=1  
>time ./test |tee output.txt (rather USE SLURM!!!)
```

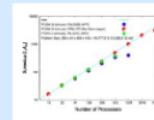
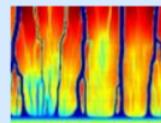
5. vary export OMP_NUM_THREADS=1,2,4,8,...

6. have a look at the profiles (profile_0000.dat)

Example: PETSc

<https://www.mcs.anl.gov/petsc/>

PETSc



Portable, Extensible Toolkit for
Scientific Computation

- [Home](#)
- [Download](#)
- [Features](#)
- [Documentation](#)
 - [Manual pages and Users](#)
 - [Manual](#)
 - [Citing PETSc](#)
 - [Tutorials](#)
 - [Installation](#)
 - [SAWs](#)
 - [Changes](#)
 - [BugReporting](#)
 - [CodeManagement](#)
 - [FAQ](#)
 - [License](#)
 - [Applications/Publications](#)
 - [Miscellaneous](#)
 - [External Software](#)
 - [Developers Site](#)

SEARCH

News: [PETSc User Meeting](#), June 28-30, 2016

The current version of PETSc is 3.7; released April 25, 2016.

- PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and [GPUs through CUDA or OpenCL](#), as well as hybrid MPI-GPU parallelism.
- [Scientific applications](#) that use PETSc
 - [Features](#) of the PETSc libraries (and a recent [podcast](#))
 - [Linear system solvers](#) accessible from PETSc
 - Related packages that use PETSc
 - [MOOSE - Multiphysics Object-Oriented Simulation Environment](#) finite element framework, built on top of libMesh and PETSc
 - [SLEPc - Scalable Library for Eigenvalue Problems](#)
 - [COOLFluiD - CFD, plasma and multi-physics simulation package](#)
 - [Fluidity - a finite element/volume fluids code](#)
 - [OpenFVM - finite volume based CFD solver](#)
 - [OOFEM - object oriented finite element library](#)
 - [libMesh - adaptive finite element library](#)
 - [FEniCS - sophisticated Python based finite element simulation package](#)
 - [Firedrake - sophisticated Python based finite element simulation package](#)
 - [DEAL_II - sophisticated C++ based finite element simulation package](#)
 - [PHAMM - The Parallel Hierarchical Adaptive MultiLevel Project](#)
 - [Chaste - Cancer, Heart and Soft Tissue Environment](#)
 - [PyClaw - A massively parallel, high order accurate, hyperbolic PDE solver](#)
 - [PetIGA - A framework for high performance Isogeometric Analysis](#)
 - [Python Bindings](#)
 - [petsc4py](#) from Lisandro Dalcin at CIMEC
 - [Elefant](#) from the SML group at NICTA
 - [Java Bindings](#)
 - [jpetsctao](#) from Hannes Sommer (this does not appear to be functional any longer)
 - [Packages](#) that PETSc can optionally use

PETSc is developed as [open-source](#), requests and [contributions](#) are welcome.

What is PETSc?

PETSc, the Portable Extendible Toolkit for Scientific Computation, is a large powerful library, mostly concerned with linear and non-linear system of equations that arise from discretized PDEs.

PETSc can be used as a **library in the traditional sense**, where you use some **high level functionality**, such as **solving a non-linear system** of equations, in your program.

However, it **can also be used as a toolbox**, to compose your own numerical applications using low-level tools.

- Linear system solvers (sparse/dense, iterative/direct)
- Non-linear system solvers
- Tools for distributed matrices
- Support for profiling, debugging, graphical output

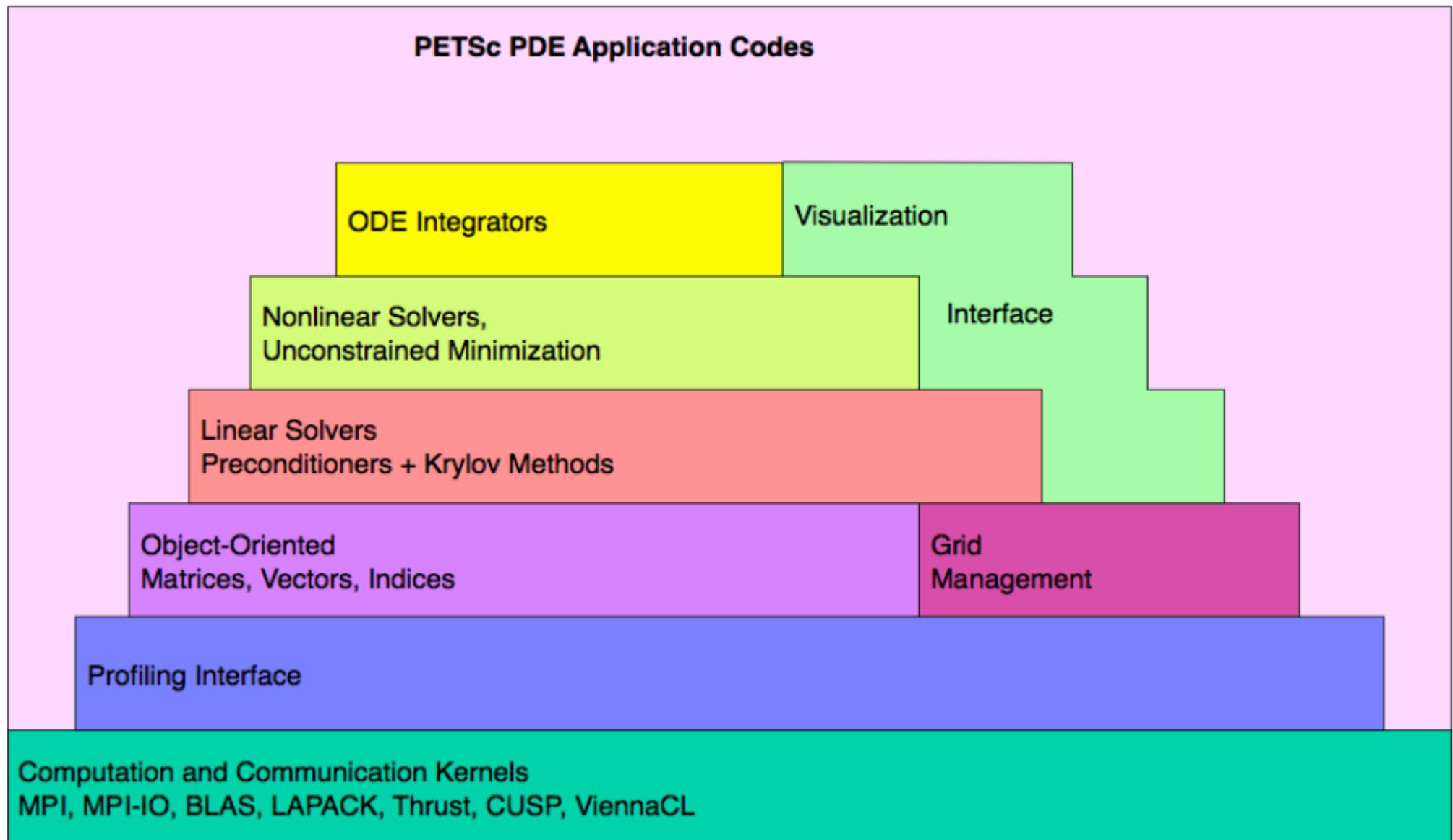
The basic functionality of PETSc can be extended through external packages:

- Dense linear algebra: Scalapack
- Grid partitioning software: ParMetis, Jostle, Chaco, Party
- ODE solvers: PVODE
- Eigenvalue solvers (including SVD): SLEPc
- Optimization: TAO

Why use PETSc?

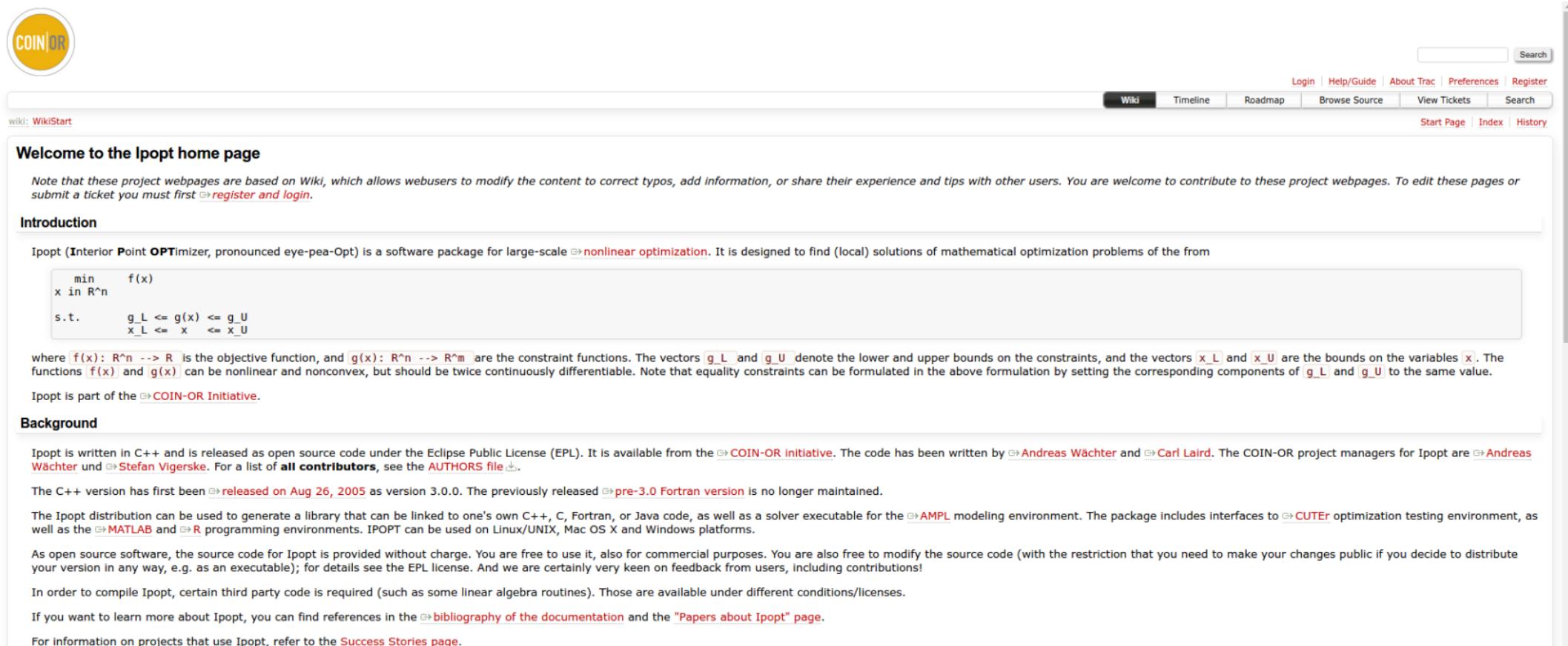
- Write robust, scalable MPI codes to solve PDEs, without writing much MPI code yourself.
- Use a combinatorial explosion of solvers, configurable at runtime.
- Run your code essentially anywhere, from your laptop up to Midway or other large clusters.
- Configure with a huge number of external packages (including external linear solvers).
- Excellent support and community.

PETSc Components



Example: Optimization

<https://projects.coin-or.org/Ipopt> – we used it in one of the projects



The screenshot shows the COIN-OR Trac project page for Ipopt. At the top left is the COIN-OR logo. The header includes links for Login, Help/Guide, About Trac, Preferences, and Register. Below the header are links for Wiki, Timeline, Roadmap, Browse Source, View Tickets, Start Page, Index, and History. A search bar is also present.

The main content area starts with a section titled "Welcome to the Ipopt home page". It contains a note about contributing to the project via the wiki and a registration/login requirement. Below this is an "Introduction" section describing Ipopt as an interior point optimizer for nonlinear optimization problems. It includes a mathematical formulation:

```

min      f(x)
x in R^n

s.t.      g_L <= g(x) <= g_U
          x_L <= x <= x_U
  
```

It explains the components of the optimization problem: $f(x)$ is the objective function, $g(x)$ are constraint functions, and x are variables. It notes that $f(x)$ and $g(x)$ can be nonlinear and nonconvex, but twice continuously differentiable. Equality constraints are handled by setting corresponding components of g_L and g_U to the same value.

The "Background" section details the project's history, mentioning its release under the Eclipse Public License (EPL), its availability from the COIN-OR initiative, and its contributors like Andreas Wächter and Carl Laird. It also notes the C++ version was released on Aug 26, 2005, and the pre-3.0 Fortran version is no longer maintained. The distribution can be used with AMPL, CUTEr, MATLAB, and R. IPOPT can be used on various platforms.

The "License" section states that the source code is provided without charge and can be modified for commercial purposes under the EPL license.

The "Compiling" section discusses the required third-party code and available conditions/licenses.

The "References" section links to the bibliography of documentation and papers about Ipopt.

The "Success Stories" section provides information on projects that use Ipopt.

Emerging hardware



GPU: NVIDIA Tesla K20c

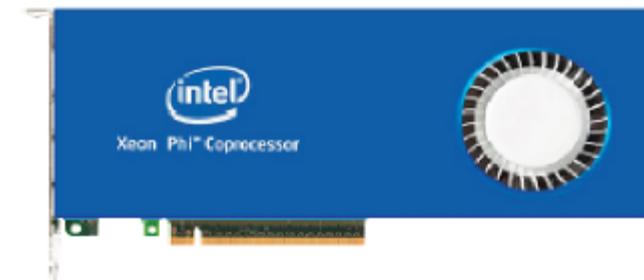
Kepler GK110, 28 nm

13 mp × 192 cores @ 0.71 GHz

5 GB GDDR5 @ 2.6 GHz

225W

→ Devices can have $O(\text{Teraflops})$



MIC: Intel Xeon Phi 3120A

Knights Corner (KNC), 22 nm

57 cores @ 1.1 GHz

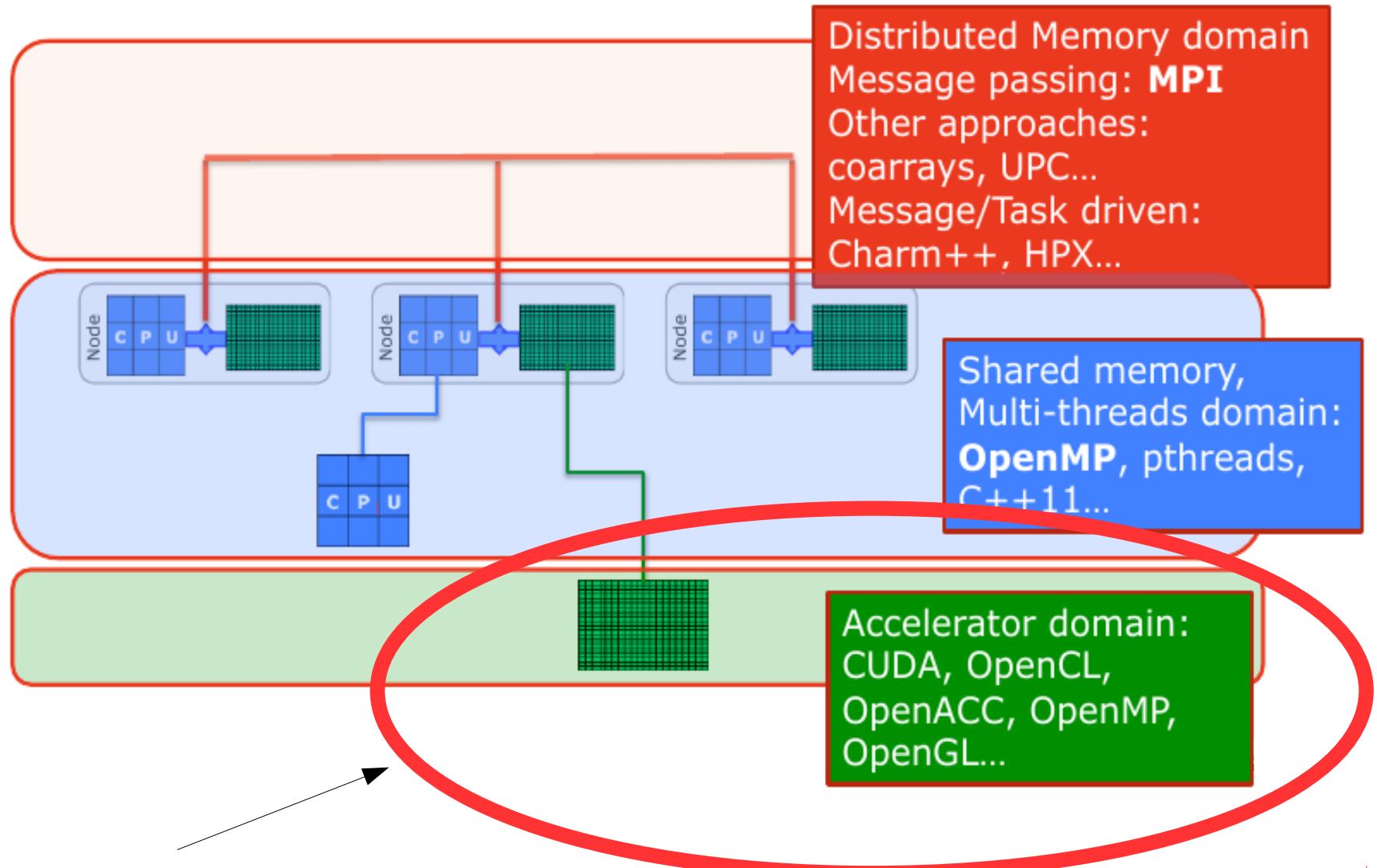
6GB GDDR5 @ 1.1 GHz

300W

up to 4 threads per core

512-bit vectorization (AVX-512)

Overall picture of programming models



Our focal point

(Slide from C. Gheller)

Why do we Need Co-processors/or “Accelerators” on HPC?

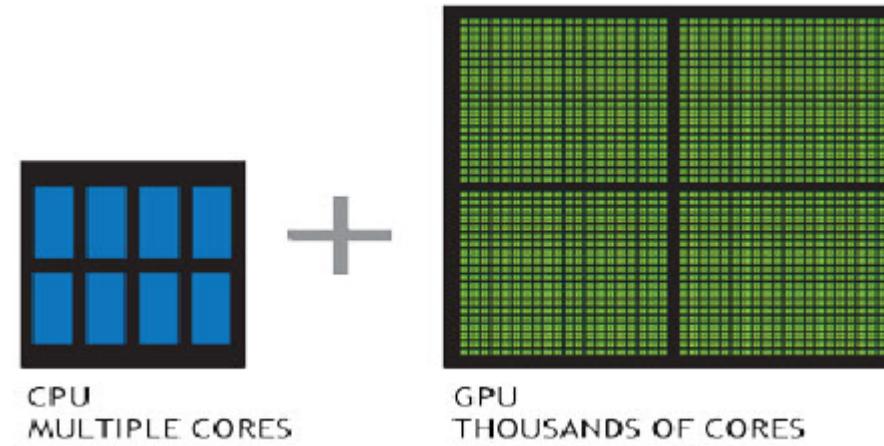
- In the past, computers got faster by increasing the **clock frequency** of the core, but this has now reached its limit mainly due to **power requirements** and heat dissipation restrictions (unmanageable problem).
- Today, processor cores are not getting any faster, but instead the number of cores per chip increases.
- On HPC, we need a chip that can provide higher computing performance at lower energy.

General Purpose GPU

- Graphics Processing Unit (GPU):
 - Hardware designed for output to display.
- **General Purpose computing on GPUs (GPGPU):**
 - Use GPUs for **non-graphics tasks**, e.g. physics simulation, signal processing, computational geometry, computer vision, database management, computational biology, computational finance
- GPUs evolved into a very flexible and powerful processor:
 - It's programmable using high-level languages

Solution

- The actual solution is a heterogeneous system containing both CPUs and “accelerators”, plus other forms of parallelism such as vector instruction support.
- Widely accepted that heterogeneous systems with accelerators deliver the highest performance and energy efficient computing in HPC.
- Today – the accelerated computing is revolutionising HPC.

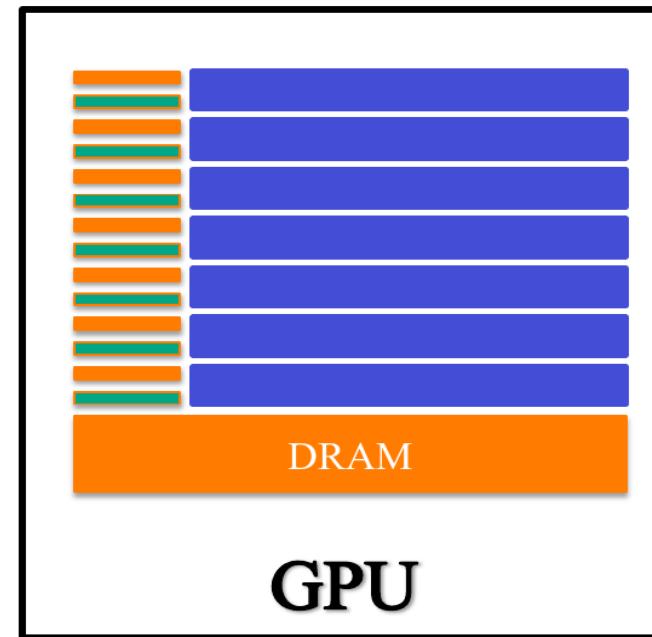
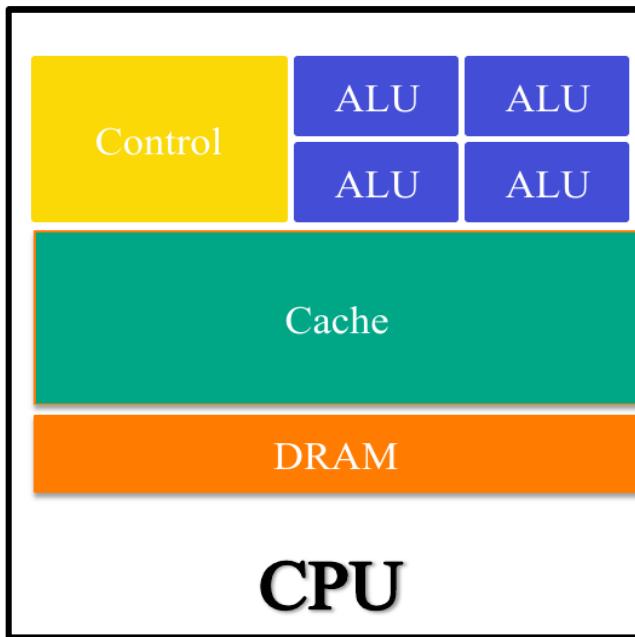


Top 500 – June 2018 (het. systems)

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,282,544	122,300.0	187,659.3	8,806
2	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
3	Sierra - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/NNSA/LLNL United States	1,572,480	71,610.0	119,193.6	
4	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	AI Bridging Cloud Infrastructure (ABCi) - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan	391,680	19,880.0	32,576.6	1,649
6	Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
7	Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
8	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
9	Trinity - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,968	14,137.3	43,902.6	3,844
10	Cori - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939

CPU vs. GPU

- Specialized for compute-intensive, highly-parallel computation, i.e. graphic output.
- Evolution pushed by gaming industry.
- CPU: large die area for control and caches.
- GPU: large die area for data processing.

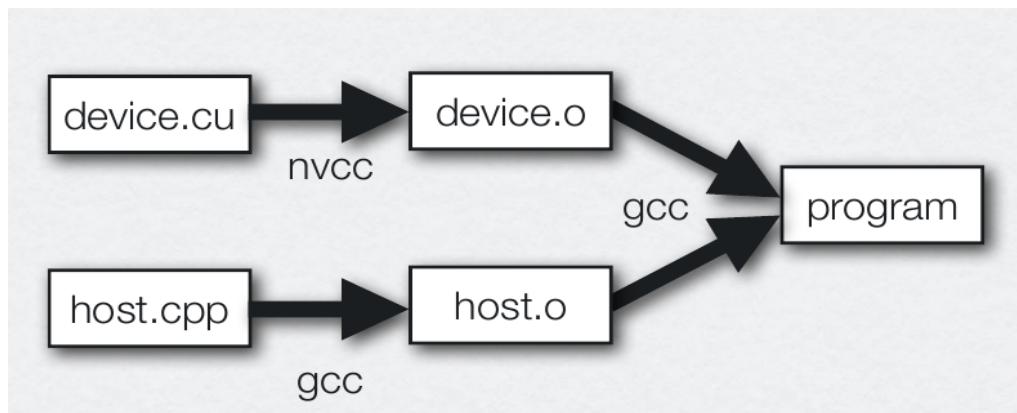


Programming GPUs

- CUDA: Nvidia proprietary API, works only on Nvidia GPUs.
- OpenCL: open standard for heterogeneous computing.
- OpenACC: open standard based on compiler directives.

Nvidia CUDA

- Compute Unified Device Architecture (CUDA)
- C extension to write GPU code, support for C++
- Only supported by Nvidia GPUs
- Code compilation (nvcc) and linking:



```
device.cu
__global__ void kernel()
{
    // do something
}

host.cpp
int main()
{}
```

OpenACC

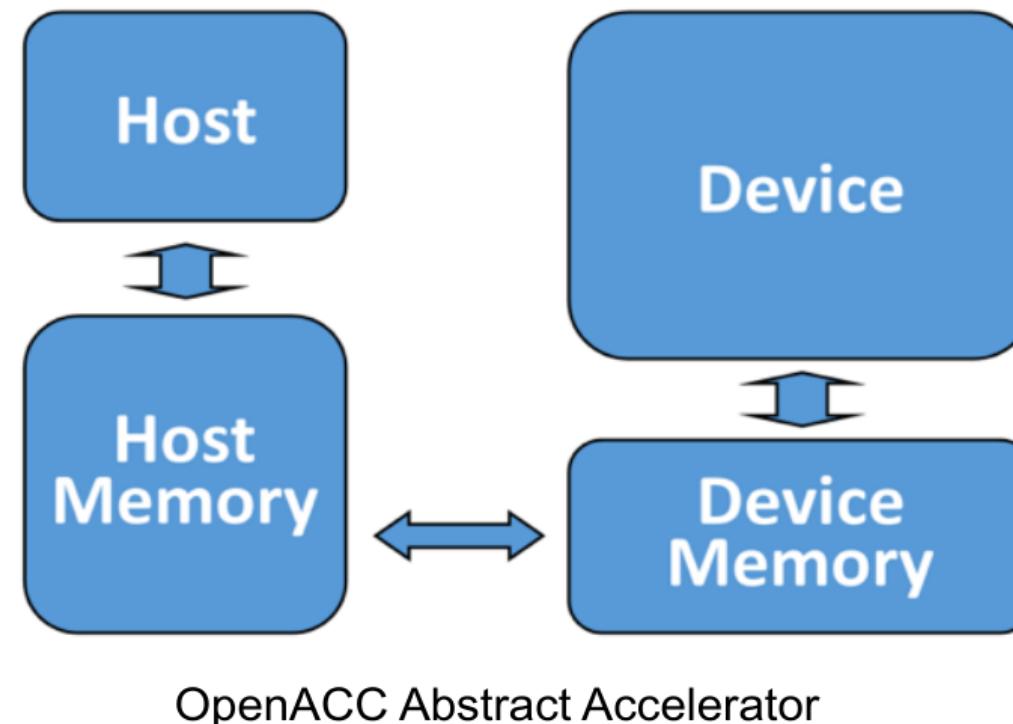
- Programming with CUDA can be more difficult than writing SPMD (i.e., MPI) applications.

- **OpenACC (Open Accelerators)**

- Developed by Cray, CAPS, Nvidia, and PGI.
- Most recent specification: 2.5 (November 2015).
- Similar to OpenMP.
- High-level of abstraction.
- OpenACC members are also part of the OpenMP language committee.
- Compiler support from Cray, PGI, and CAPS.
- Experimental support for OpenACC in GCC/5.1.

OpenACC

- The **OpenACC API is a set of compiler directives** for offloading work to accelerators.
- For many systems, there will be a CPU host and GPU accelerator.
- **OpenACC will handle any accelerator memory management and the transfer of data.**



Directives

- Like OpenMP, OpenACC is primarily programmed using directives.
- Lower-level programming models like CUDA perform better for certain optimizations (i.e. abstraction penalty).

C/C++

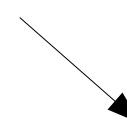
#pragma acc directive-name [clause-list] new-line

Scope is the following block of code

Fortran

!\$acc directive-name [clause-list] new-line

Scope is until **!\$acc end** directive name



```
9 #pragma acc parallel loop
10   for (i=0; i<N; i++) {
11     y[i] = 0.0;
12     x[i] = (double) (i+1);
13   }
14 
```

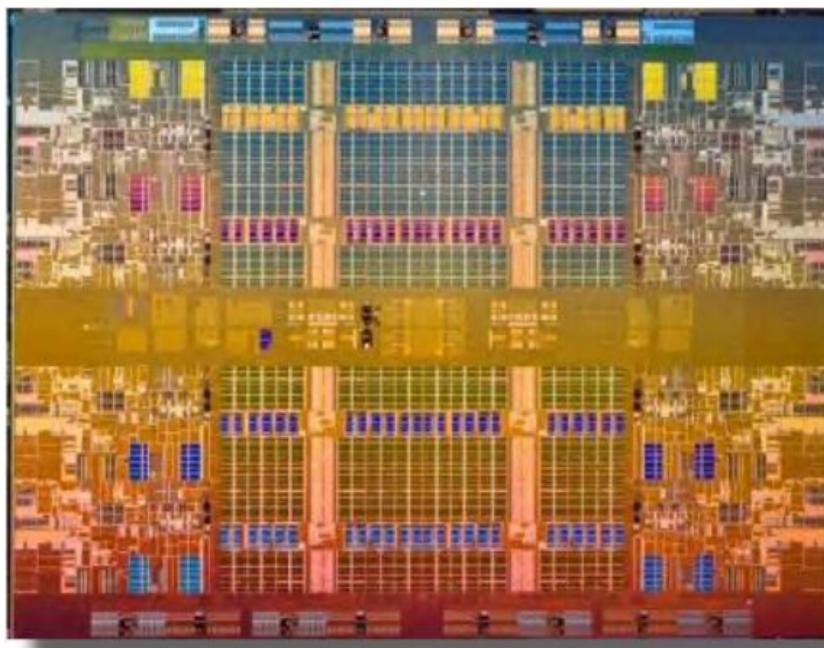
Intel Xeon Phi Products

- The first product was released in 2012 named **Knights Corner (KNC)** which is the first architecture supporting **512 bit vectors**.
- The 2nd generation released last week named **Knights Landing (KNL)** also support 512bit vectors with a new instruction set called Intel Advanced Vector Instructions 512 (Intel AVX-512).
- KNL has a peak performance of **6 TFLOP/s in single precision** ~ 3 times what KNC had, due to 2 **vector processing units (VPUs)** per core, doubled compared to the KNC.
 - Each VPU operates independently on 512-bit vector registers, which can hold 16 single precision or 8 double precession floating-point numbers.



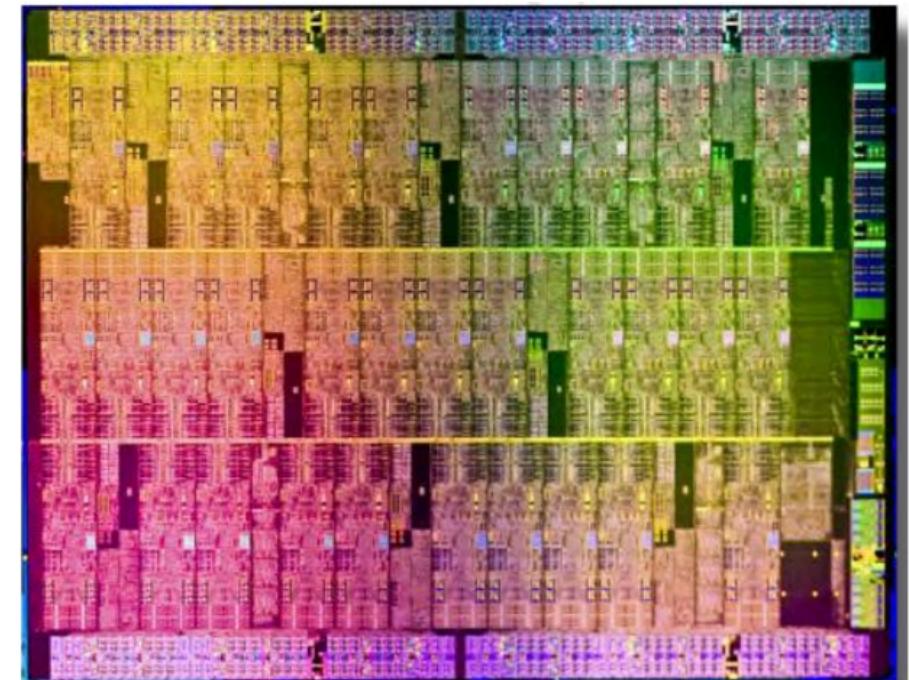
Multi-core vs. Many-core

Parallel Programming and Optimization with Intel Xeon Phi Coprocessors, Colfax 2013
<http://www.colfaxintl.com/nd/xeonphi/book.aspx>



Multi-core Intel Xeon processor

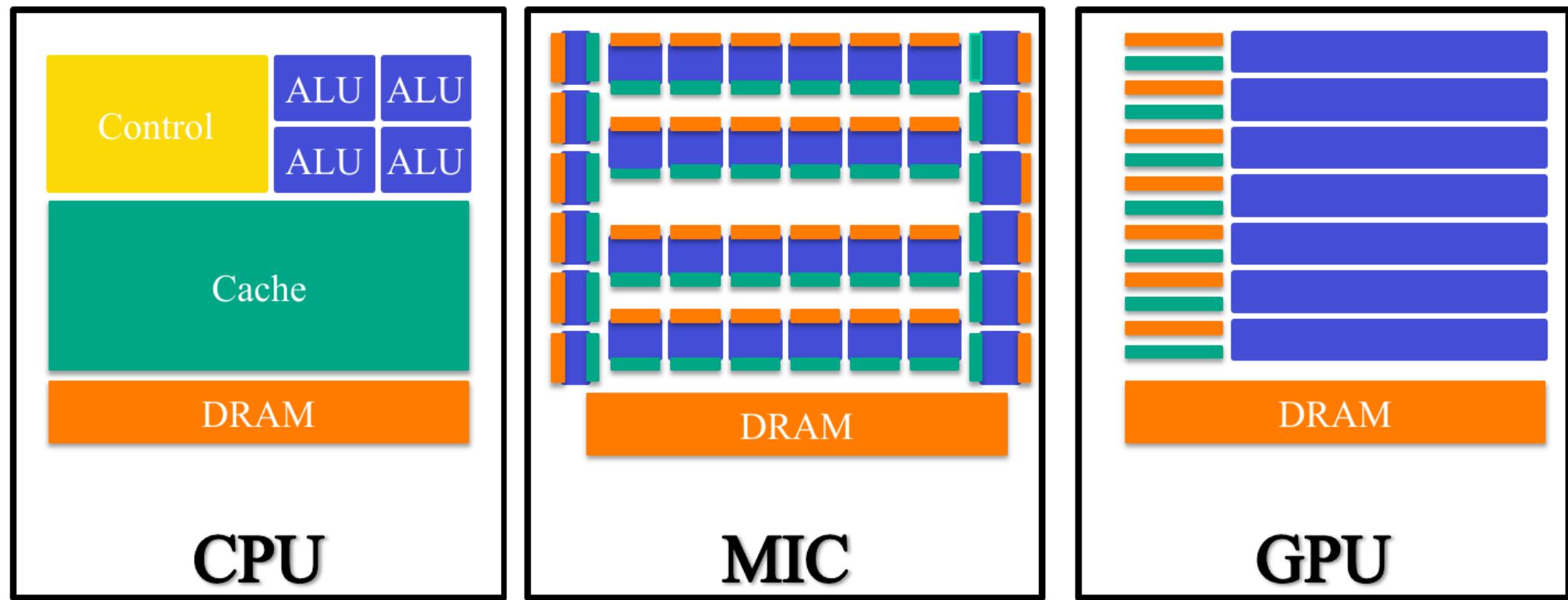
- ~ 16 physical cores ~ 3 GHz
- e.g Intel Sandy-Bridge 32 nm
- (AVX) 256-bit vector registers



Many-core Intel Xeon Phi coprocessor

- ~ 61 cores (244) ~ 1 GHz
- 22 nm
- 512-bit vector registers

Architecture comparison



General-purpose
architecture

Power-efficient
Multiprocessor X86
design architecture

**Massively data
parallel**

Advantages of MIC

- **Retains programmability and flexibility** of standard x86 architecture.
- No need to learn a new complicated language like CUDA or OpenCL.
- Offers possibilities we miss on GPUs: Login onto the system, watching and controlling processes via top, kill etc. like on a Linux host.
- **Allows many different parallel programming models like OpenMP, MPI, and Intel Threading Building Blocks (TBB).**
- Offers standard math-libraries like Intel MKL.
- **Supports whole Intel tool chain**, e.g. Intel C/C++ and Fortran Compiler, Debugger & Intel VTune Amplifier.

High throughput computing/Cloud



Definition of cloud computing

Cloud computing enables users to consume a compute resource, such as a virtual machine (VMs), storage or an application, as a utility – just like electricity – rather than having to build and maintain computing infrastructures in house.

Cloud computing boosts several attractive benefits for end users. Three of the main benefits of cloud computing are:

- **Self-service provisioning**: End users can spin up compute resources for almost any type of **workload on demand**. This eliminates the traditional need for IT administrators to provision and manage compute resources.
- **Elasticity**: Users can scale up as computing needs increase and scale down again as demands decrease. This eliminates the need for massive investments in local infrastructure which may or may not remain active.
- **Pay per use**: Compute resources are measured at a granular level, allowing users to pay only for the resources and workloads they use.

Examples: Amazon's EC2 compute service; Amazon S3 storage service; Apple's MobileMe (storage, file sharing, etc.); Flickr, Netflix, YouTube; etc.

High throughput computing (HTC)

- **High-throughput computing (HTC)** is a computer science term to describe the use of many computing resources over long periods of time to accomplish a computational task.
- Using distributed computing (potentially grid computing) to **enable lots of jobs to be scheduled to available resources to complete as fast as possible**.
- The term was popularized e.g. by Condor project (U Wisconsin).
- Examples: Condor, World Community Grid, LHC project, Open Science Grid, etc.
- Goal: to integrate multiple computing systems to enable large numbers of tasks to be scheduled and completed as rapidly as possible. **Resources: can be centrally managed servers (clusters, clouds) and/or distributed PCs.**

Example – HTCondor

<https://research.cs.wisc.edu/htcondor/>

<http://chtcs.cs.wisc.edu/>



Google™ Custom Search

Computing with HTCondor™

Our goal is to develop, implement, deploy, and evaluate mechanisms and policies that support [High Throughput Computing \(HTC\)](#) on large collections of distributively owned computing resources. Guided by both the technological and sociological challenges of such a computing environment, the [Center for High Throughput Computing](#) at UW-Madison has been building the open source [HTCondor distributed computing software](#) (pronounced "atch-tee-condor") and related technologies to enable scientists and engineers to increase their computing throughput.

Grid Computing Definition*

*https://en.wikipedia.org/wiki/Grid_computing

Grid computing is the most distributed form of parallel computing.

It makes use of computers communicating over the Internet to work on a given problem.

Because of the low bandwidth and extremely **high latency** available on the Internet, distributed computing typically deals only with **embarrassingly parallel** problems.

Many distributed computing applications have been created, of which **SETI@home, LHC@home and Folding@home** are the best-known examples.

Most grid computing applications use **middle-ware**, software that sits between the operating system and the application to **manage network resources** and standardize the software interface.

Often, distributed computing software makes use of "spare cycles", performing computations at times when a computer is idling.

Example – LHC@home

<http://lhcatome.web.cern.ch/>

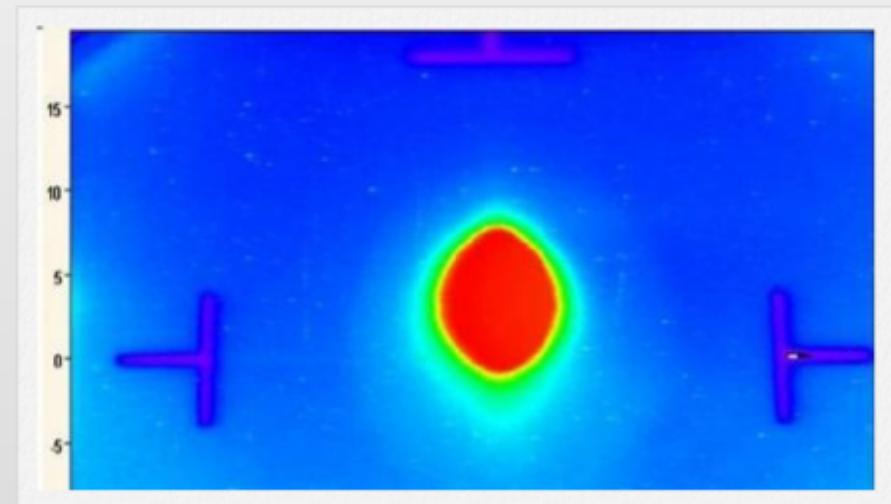
The screenshot shows the LHC@home website on a CERN Accelerating science page. The header includes the CERN logo and links for 'Sign in' and 'Directory'. The main content area features the LHC@home logo with the tagline 'Volunteer computing for the LHC'. A search bar with a 'Search' button is also present. Below the header is a navigation bar with links for 'HOME', 'ABOUT', 'PROJECTS', 'JOIN US!', 'HELP & FAQ', and 'CONTACT'.

Sixtrack

Create better beams!

Magnetic imperfections, electromagnetic wake, even gravity - so many things can destabilise a proton beam. Help us create better beams!

[Tell me more](#)



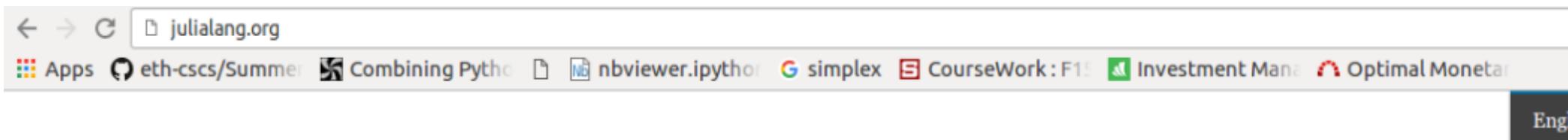
The value of HTC

- Majority of computational science is performed on workstations/PCs
 - HPC systems not needed for all tasks.
 - Many tasks need to be repeated a lot.
- Running simulations to explore a parameter space
 - Running simulations on different data sets.
 - Analyzing experimental results that ongoing/repeated.
- HTC tools enable this, sometimes from the desktop (e.g. Condor)
- Think of Big Data analytics... (<http://lhcathome.web.cern.ch/>)

Trending topics

<http://julialang.org/learning/>

[https://en.wikipedia.org/wiki/Julia_\(programming_language\)](https://en.wikipedia.org/wiki/Julia_(programming_language))



[home](#) [source](#) [downloads](#) [docs](#) [packages](#) [blog](#) [community](#) [learning](#) [teaching](#) [publications](#) [GSoC](#) [juliacon](#)

Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Julia's Base library, largely written in Julia itself, also integrates mature, best-of-breed open source C and Fortran libraries for linear algebra, random number generation, signal processing, and string processing. In addition, the Julia developer community is contributing a number of external packages through Julia's built-in package manager at a rapid pace. IJulia, a collaboration between the Jupyter and Julia communities, provides a powerful browser-based graphical notebook interface to Julia.

Julia programs are organized around multiple dispatch; by defining functions and overloading them for different combinations of argument types, which can also be user-defined. For a more in-depth discussion of the rationale and advantages of Julia over other systems, see the following highlights or read the introduction in the online manual.

IEEE – The Top Programming Languages 2018

*<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1
11. Matlab		72.8
12. Scala		72.1
13. Ruby		71.4
14. HTML		71.2
15. Arduino		69.0
16. Shell		66.1
17. Perl		57.4
18. Swift		53.9
19. Processing		53.1
20. Objective-C		50.5
21. Lua		49.8
22. Fortran		49.5
23. SQL		49.3
24. Haskell		48.6
26. Visual Basic		45.1
27. Cuda		43.0
28. Rust		41.8
29. Verilog		41.2
30. D		40.6
31. Delphi		38.7
32. Julia		35.1
33. Lisp		33.3
34. Prolog		33.2
35. LabView		32.7
36. Erlang		26.9
37. SAS		25.6
38. Clojure		25.6
39. Cobol		24.6
40. ABAP		22.8
41. TCL		21.9
42. Ada		20.9
43. Scheme		18.8
44. J		18.1
45. Ocaml		14.4
46. Ladder Logic		11.5
47. Actionscript		0.6
48. Forth		0.0

Apache Hadoop

hadoop.apache.org

Apache > Hadoop >



Search with Apache Solr Search Last Published: 10/11/2016 15:50:42

About

- Welcome
- What Is Apache Hadoop...
- Getting Started ...
- Download Hadoop
- Who Uses Hadoop?...
- News
- Releases
- Release Versioning
- Mailing Lists
- Issue Tracking
- Who We Are?
- Who Uses Hadoop?
- Buy Stuff
- Sponsorship
- Thanks
- Privacy Policy
- Bylaws
- Committer criteria
- License

Documentation

Related Projects

Welcome to Apache™ Hadoop®!

What Is Apache Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

- Hadoop Common:** The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™):** A distributed file system that provides high-throughput access to application data.
- Hadoop YARN:** A framework for job scheduling and cluster resource management.
- Hadoop MapReduce:** A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

Apache Hadoop (2)

https://en.wikipedia.org/wiki/Apache_Hadoop

<http://hadoop.apache.org/>

https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm

- Apache Hadoop is an open-source software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.
- All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework.
- The core of Apache Hadoop consists of a storage part, known as **Hadoop Distributed File System (HDFS)**, and a processing part called **MapReduce**.
- **Hadoop splits files into large blocks and distributes them across nodes in a cluster.**
- To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed.
- This approach takes advantage of data locality – nodes manipulating the data they have access to – to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Quantum Computing – industry explosion

The New York Times

Microsoft Spends Big to Build a Computer Out of Science Fiction

By JOHN MARKOFF NOV. 20, 2016

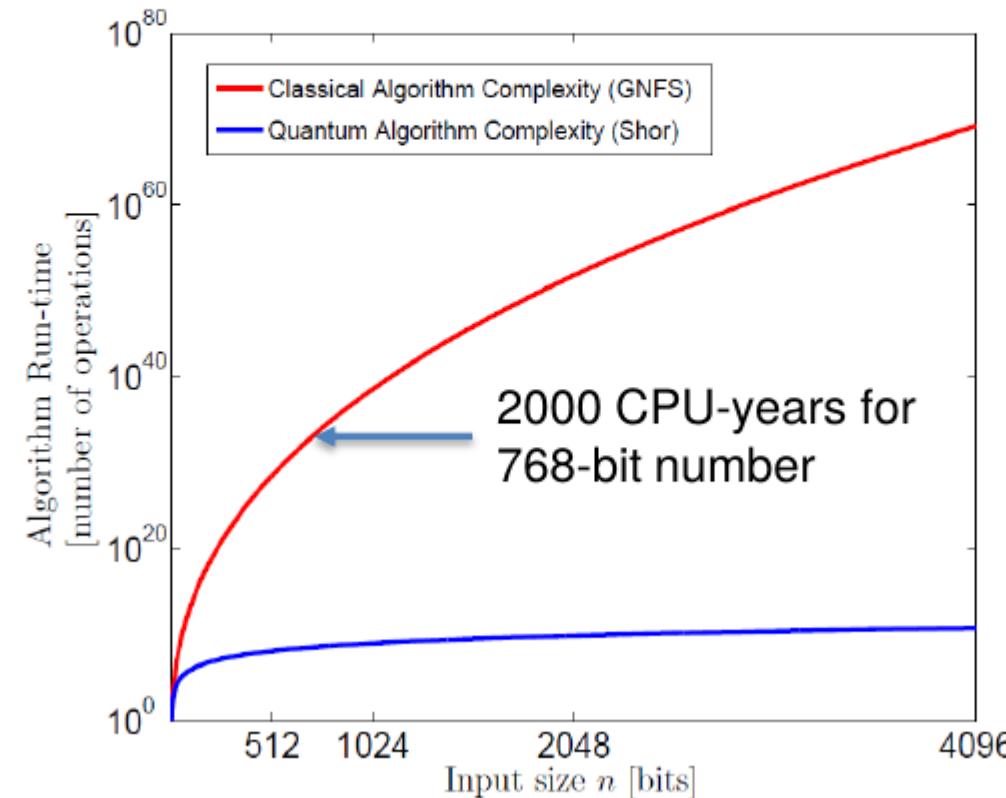
Companies building quantum computing hardware:



3

The promise of quantum computing

Vastly faster computation for some very specific problems.



What is a quantum computer?

A **computer** that fundamentally takes advantage of the laws of **quantum mechanics** to solve problems.

(Quantum mechanics is the theory of how physical systems behave, and is typically relevant only when the system is very small.)

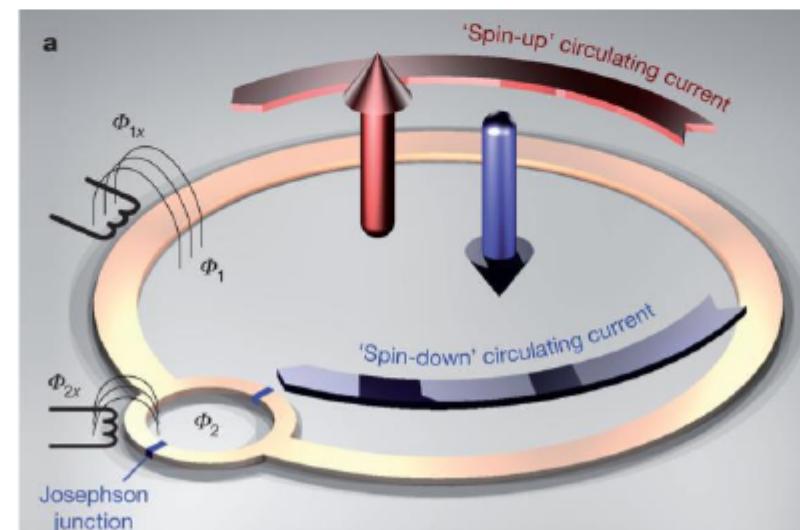


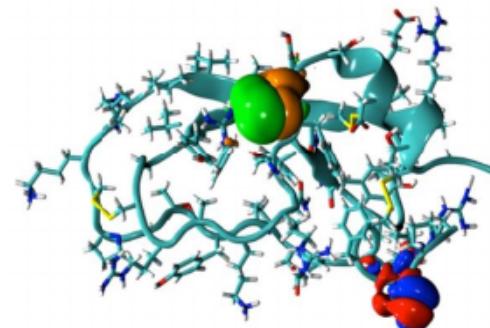
Image credit: M.W. Johnson, et al. *Nature* 473, 194 (2011)

What is a quantum computer good for?

There are two major applications of quantum computing that the community is very convinced about. There are several others that are more speculative.

Quantum Chemistry

Calculation of chemical properties from first principles.



17% of Oak Ridge National Lab **supercomputer time** is spent on quantum chemistry (source: Juerg Hutter, 2011)

Prime Factoring

Breaking a product of two prime numbers into its prime factors (main use: breaking modern public-key cryptography systems)

$$15 = 3 \times 5$$

$$642469 = 601 \times 1069$$

Substantial resources are required for classical algorithms: **2000 CPU-years** for **768-bit** number⁷

Moving forward

- Quantum computer hardware development now has strong industrial backing (Google, Microsoft, IBM, etc.)
- Google is expected to demonstrate “quantum supremacy” for the first time within the next year using a 49-qubit machine
- Google is aiming to reach 1 million physical qubits by 2027
- IBM will release a 17-qubit machine this year (and has 5-qubit machine available already)
- There are two well-established areas in which QC will give an advantage (chemistry and cryptography breaking)
- Discovering other areas is an active area of research

Wrap-up (1)

Lecture 1

1. Introduction to parallel and high-performance computing.
2. Introduction to a (discrete state) dynamic programming code in C++.

Lecture 2

1. Introduction to OpenMP (shared-memory parallelization).
2. A shared memory parallel dynamic programming code.

Wrap-up (2)

Lecture 3

1. Introduction to MPI (distributed-memory parallelization).
2. A distributed memory parallel dynamic programming code.

Lecture 4:

1. Hybrid parallelism – OpenMP & MPI.
2. A hybrid parallel dynamic programming code.
3. Advanced topics.



KEEP
CALM
AND

ACCEPT THE FACT
THAT IT'S OVER